9.4

Securing IBM MQ

Note

Before using this information and the product it supports, read the information in "Notices" on page 677.

This edition applies to version 9 release 4 of IBM® MQ and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

Securing IBM MQ

Security is an important consideration for both developers of IBM MQ applications, and for IBM MQ system administrators. As an absolute minimum, you should ensure that all hardware and software inside the secure zone and on operator workstations are within their support lifecycle, are up-to-date with mandatory software updates, and have security updates promptly applied.

Related reference

IBM Security Vulnerability Management

z/OS IBM Z and LinuxOne Security Portal

Security overview

This collection of topics introduces the IBM MQ security concepts.

Security concepts and mechanisms, as they apply to any computer system, are presented first, followed by a discussion of those security mechanisms as they are implemented in IBM MQ.

The commonly accepted aspects of security are as follows:

- "Identification and authentication" on page 7
- "Authorization" on page 9
- "Auditing" on page 9
- "Confidentiality" on page 10
- "Data integrity" on page 10

Security mechanisms are technical tools and techniques that are used to implement security services. A mechanism might operate by itself, or with others, to provide a particular service. Examples of common security mechanisms are as follows:

- "Cryptography" on page 11
- "Message digests and digital signatures" on page 13
- "Digital certificates" on page 13
- "Public Key Infrastructure (PKI)" on page 17

When you are planning an IBM MQ implementation, consider which security mechanisms you require to implement those aspects of security that are important to you. For information about what to consider after you have read these topics, see "Planning for your security requirements" on page 85.

Identification and authentication

Identification is the ability to identify uniquely a user of a system or an application that is running in the system. *Authentication* is the ability to prove that a user or application is genuinely who that person or what that application claims to be.

For example, consider a user who logs on to a system by entering a user ID and password. The system uses the user ID to identify the user. The system authenticates the user at the time of logon by checking that the supplied password is correct.

Identification and authentication in IBM MQ

When an application connects to IBM MQ, a user identity is always associated with the connection. The user identity is initially the Operating System user ID that is associated with the application process. This identity is often sufficient for locally bound applications that are hosted on the same system as the queue manager. However, the queue manager can also authenticate and modify the identity that is associated with the connection in several ways. Authenticating the identity that is associated with a connection is important when client applications that cannot necessarily be trusted connect to a queue manager over a network.

The identity that is associated with an application connection to an IBM MQ queue manager can be established by using any of the following mechanisms:

- When an application connects to a queue manager, it can provide a user ID and password. The queue manager validates the credentials based on its configuration. For example, the user ID and password can be passed to the queue manager's Operating System, or and LDAP server, to be authenticated.
- V 9.4.0 From IBM MQ 9.3.4, an application can also supply an authentication token that it obtains from an external authentication server. For more information about authentication tokens, see "Working with authentication tokens" on page 318.
- A client channel can be configured to use TLS mutual authentication, if it is configured with a valid digital certificate. TLS authentication can be combined with a channel authentication (CHLAUTH) rule to associate an appropriate user ID with the connection. For more information, see "How TLS provides identification, authentication, confidentiality, and integrity" on page 20,
- Channel authentication (CHLAUTH) rules can override the identity based on information about the connection. For example, a channel authentication rule can set the user ID associated with a connection based on the IP address of the client.
- Custom exit code can set an identity based on any criteria that you choose.

Identity and authentication are also applicable to channels between two queue managers. These channels are known as message channels. When a message channel starts, the message channel agent (MCA) at each end of the channel can authenticate its partner. This technique is known as *mutual authentication*. For the sending MCA, it provides assurance that the partner it is about to send messages to is genuine. Similarly, the receiving MCA is assured that it is about to receive messages from a genuine partner.

When an identity has been established, and authenticated if required, it is used by IBM MQ in several ways:

- Importantly, by default, any subsequent "Authorization" on page 9 checks are made using this identity. For example, if an application attempts to put a message on a queue, the queue manager confirms that the identity that is associated with the application has 'put' authorization on the queue object.
- In addition, every message can contain *message context* information. This information is held in the message descriptor (MQMD). The queue manager can automatically generate the message context when an application puts the message to a queue. Alternatively, the application can supply the message context if the user ID associated with the application is authorized to do so. This context information in a message gives the application that receives the message information about the originator of the message. It contains, for example, the name of the application that put the message and the user ID associated with the application.

Non-repudiation

The overall goal of the non-repudiation service is to be able to prove that a particular message is associated with a particular individual.

The *non-repudiation* service can be viewed as an extension to the identification and authentication service. In general, non-repudiation applies when data is transmitted electronically; for example, an order to a stock broker to buy or sell stock, or an order to a bank to transfer funds from one account to another.

The non-repudiation service can contain more than one component, where each component provides a different function. If the sender of a message ever denies sending it, the non-repudiation service with *proof of origin* can provide the receiver with undeniable evidence that the message was sent by that particular individual. If the receiver of a message ever denies receiving it, the non-repudiation service with *proof of delivery* can provide the sender with undeniable evidence that the message was received by that particular individual.

In practice, proof with virtually 100% certainty, or undeniable evidence, is a difficult goal. In the real world, nothing is fully secure. Managing security is more concerned with managing risk to a level that is acceptable to the business. In such an environment, a more realistic expectation of the non-repudiation service is to be able to provide evidence that is admissible, and supports your case, in a court of law.

Non-repudiation is a relevant security service in an IBM MQ environment because IBM MQ is a means of transmitting data electronically. For example, you might require contemporaneous evidence that a particular message was sent or received by an application associated with a particular individual.

IBM MQ with Advanced Message Security does not provide a non-repudiation service as part of its base function. However, this product documentation does contain suggestions on how you might provide your own non-repudiation service within an IBM MQ environment by writing your own exit programs.

Authorization

Authorization protects critical resources in a system by limiting access only to authorized users and their applications. It prevents the unauthorized use of a resource or the use of a resource in an unauthorized manner.

Authorization in IBM MQ

You can use authorization to limit what particular individuals or applications can do in your IBM MQ environment.

Here are some examples of authorization in an IBM MQ environment:

- Allowing only an authorized administrator to issue commands to manage IBM MQ resources.
- Allowing an application to connect to a queue manager only if the user ID associated with the application is authorized to do so.
- Allowing an application to open only those queues that are necessary for its function.
- Allowing an application to subscribe only to those topics that are necessary for its function.
- Allowing an application to perform only those operations on a queue that are necessary for its function. For example, an application might need only to browse messages on a particular queue, and not to put or get messages.

For more information about how to set up authorization, see “Planning authorization” on page 89 and the associated sub-topics.

Auditing

Auditing is the process of recording and checking events to detect whether any unexpected or unauthorized activity has taken place, or whether any attempt has been made to perform such activity.

Auditing in IBM MQ

IBM MQ can issue event messages to record that unusual activity has taken place.

Here are some examples of auditing in an IBM MQ environment:

- An application attempts to open a queue that it is not authorized to open. An instrumentation event message is issued. By inspecting the event message, you discover that this attempt occurred and can decide what action is necessary.
- An application attempts to open a channel, but the attempt fails because the TLS connection is not allowed. An instrumentation event message is issued. By inspecting the event message, you discover that this attempt occurred and can decide what action is necessary.

Confidentiality

The *confidentiality* service protects sensitive information from unauthorized disclosure.

When sensitive data is stored locally, access control mechanisms might be sufficient to protect it on the assumption that the data cannot be read if it cannot be accessed. If a greater level of security is required, the data can be encrypted.

Encrypt sensitive data when it is transmitted over a communications network, especially over an insecure network such as the Internet. In a networking environment, access control mechanisms are not effective against attempts to intercept the data, such as wiretapping.

Confidentiality in IBM MQ

You can implement confidentiality in IBM MQ by encrypting messages.

Confidentiality can be ensured in an IBM MQ environment as follows:

- After a sending MCA gets a message from a transmission queue, IBM MQ uses TLS to encrypt the message before it is sent over the network to the receiving MCA. At the other end of the channel, the message is decrypted before the receiving MCA puts it on its destination queue.
- While messages are stored on a local queue, the access control mechanisms provided by IBM MQ might be considered sufficient to protect their contents against unauthorized disclosure. However, for a greater level of security, you can use Advanced Message Security to encrypt the messages stored in the queues.
- z/OS Messages stored on local queues can be encrypted at rest using z/OS® data set encryption.

  See the section, confidentiality for data at rest on IBM MQ for z/OS with data set encryption. for more information.

Data integrity

The *data integrity* service detects whether there has been unauthorized modification of data.

There are two ways in which data might be altered: accidentally, through hardware and transmission errors, or because of a deliberate attack. Many hardware products and transmission protocols have mechanisms to detect and correct hardware and transmission errors. The purpose of the data integrity service is to detect a deliberate attack.

The data integrity service aims only to detect whether data has been modified. It does not aim to restore data to its original state if it has been modified.

Access control mechanisms can contribute to data integrity insofar as data cannot be modified if access is denied. But, as with confidentiality, access control mechanisms are not effective in a networking environment.

Data integrity in IBM MQ

Data integrity can be ensured in an IBM MQ environment as follows:

- You can use TLS to detect whether the contents of a message have been deliberately modified while it was being transmitted over a network. In TLS, the message digest algorithm provides detection of modified messages in transit.

  All IBM MQ CipherSpecs provide a message digest algorithm, except for TLS_RSA_WITH_NULL_NULL, which does not provide message data integrity.

  IBM MQ detects modified messages upon receiving them; on receiving a modified message, IBM MQ an AMQ9661 error message is written to the error log and the channel stops.
- While messages are stored on a local queue, the access control mechanisms provided by IBM MQ might be considered sufficient to prevent deliberate modification of the contents of the messages.

However, for a greater level of security, you can use Advanced Message Security to detect whether the contents of a message have been deliberately modified between the time the message was put on the queue and the time it was retrieved from the queue.

If a modified message is detected, the application attempting to receive the message receives a MQRC_SECURITY_ERROR (2063) return code. If the application is using an MQGET call, the message is also moved to the SYSTEM.PROTECTION.ERROR.QUEUE queue.

Cryptographic concepts

This collection of topics describes the concepts of cryptography applicable to IBM MQ.

The term *entity* is used to refer to a queue manager, an IBM MQ MQI client, an individual user, or any other system capable of exchanging messages.

Cryptography

Cryptography is the process of converting between readable text, called *plaintext*, and an unreadable form, called *ciphertext*.

This occurs as follows:

1. The sender converts the plaintext message to ciphertext. This part of the process is called *encryption* (sometimes *encipherment*).
2. The ciphertext is transmitted to the receiver.
3. The receiver converts the ciphertext message back to its plaintext form. This part of the process is called *decryption* (sometimes *decipherment*).

The conversion involves a sequence of mathematical operations that change the appearance of the message during transmission but do not affect the content. Cryptographic techniques can ensure confidentiality and protect messages against unauthorized viewing (eavesdropping), because an encrypted message is not understandable. Digital signatures, which provide an assurance of message integrity, use encryption techniques. See "Digital signatures in SSL/TLS" on page 22 for more information.

Cryptographic techniques involve a general algorithm, made specific by the use of keys. There are two classes of algorithm:

- Those that require both parties to use the same secret key. Algorithms that use a shared key are known as *symmetric* algorithms. Figure 1 on page 12 illustrates symmetric key cryptography.
- Those that use one key for encryption and a different key for decryption. One of these must be kept secret but the other can be public. Algorithms that use public and private key pairs are known as *asymmetric* algorithms. Figure 2 on page 12 illustrates asymmetric key cryptography, which is also known as *public key cryptography*.

The encryption and decryption algorithms used can be public but the shared secret key and the private key must be kept secret.

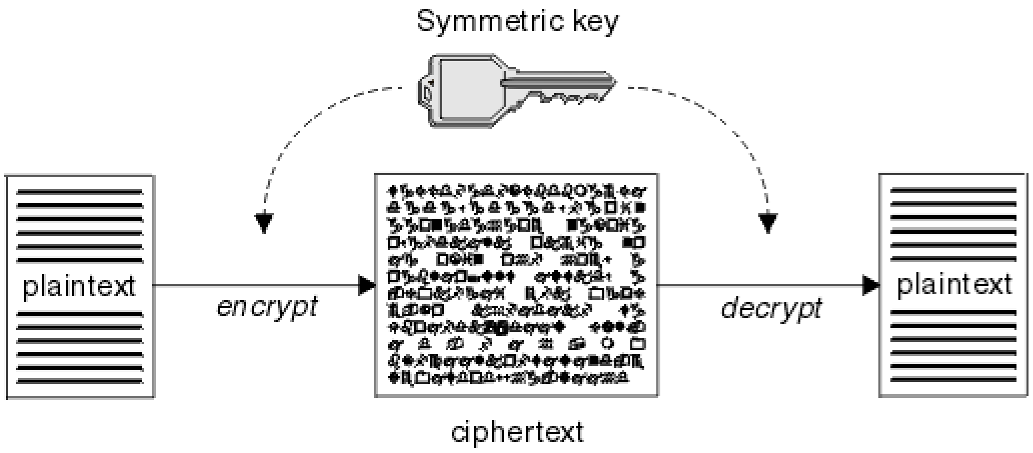


Figure 1. Symmetric key cryptography

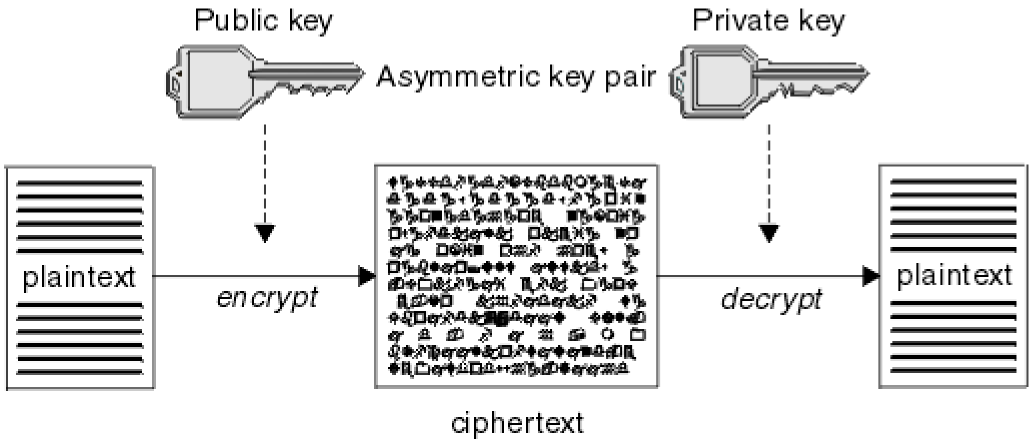


Figure 2. Asymmetric key cryptography

Figure 2 on page 12 shows plaintext encrypted with the receiver's public key and decrypted with the receiver's private key. Only the intended receiver holds the private key for decrypting the ciphertext. Note that the sender can also encrypt messages with a private key, which allows anyone that holds the sender's public key to decrypt the message, with the assurance that the message must have come from the sender.

With asymmetric algorithms, messages are encrypted with either the public or the private key but can be decrypted only with the other key. Only the private key is secret, the public key can be known by anyone. With symmetric algorithms, the shared key must be known only to the two parties. This is called the *key distribution problem*. Asymmetric algorithms are slower but have the advantage that there is no key distribution problem.

Other terminology associated with cryptography is:

Strength

The strength of encryption is determined by the key size. Asymmetric algorithms require large keys, for example:

1024 bits	Low-strength asymmetric key
2048 bits	Medium-strength asymmetric key
4096 bits	High-strength asymmetric key

Symmetric keys are smaller: 256 bit keys give you strong encryption.

Block cipher algorithm
These algorithms encrypt data by blocks. For example, the RC2 algorithm from RSA Data Security Inc. uses blocks 8 bytes long. Block algorithms are typically slower than stream algorithms.

Stream cipher algorithm
These algorithms operate on each byte of data. Stream algorithms are typically faster than block algorithms.

Message digests and digital signatures

A message digest is a fixed size numeric representation of the contents of a message. The message digest is computed by a hash function and can be encrypted, forming a digital signature.

The hash function used to compute a message digest must meet two criteria:

- It must be one way. It must not be possible to reverse the function to find the message corresponding to a particular message digest, other than by testing all possible messages.
- It must be computationally infeasible to find two messages that hash to the same digest.

The message digest is sent with the message itself. The receiver can generate a digest for the message and compare it with the digest of the sender. The integrity of the message is verified when the two message digests are the same. Any tampering with the message during transmission almost certainly results in a different message digest.

A message digest created using a secret symmetric key is known as a Message Authentication Code (MAC), because it can provide assurance that the message has not been modified.

The sender can also generate a message digest and then encrypt the digest using the private key of an asymmetric key pair, forming a digital signature. The signature must then be decrypted by the receiver, before comparing it with a locally generated digest.

Related concepts
"Digital signatures in SSL/TLS" on page 22
A digital signature is formed by encrypting a representation of a message. The encryption uses the private key of the signatory and, for efficiency, usually operates on a message digest rather than the message itself.

Digital certificates

Digital certificates protect against impersonation, certifying that a public key belongs to a specified entity. They are issued by a Certificate Authority.

Digital certificates provide protection against impersonation, because a digital certificate binds a public key to its owner, whether that owner is an individual, a queue manager, or some other entity. Digital certificates are also known as public key certificates, because they give you assurances about the ownership of a public key when you use an asymmetric key scheme. A digital certificate contains the public key for an entity and is a statement that the public key belongs to that entity:

- When the certificate is for an individual entity, the certificate is called a *personal certificate* or *user certificate*.
- When the certificate is for a Certificate Authority, the certificate is called a *CA certificate* or *signer certificate*.

If public keys are sent directly by their owner to another entity, there is a risk that the message could be intercepted and the public key substituted by another. This is known as a *man in the middle attack*. The solution to this problem is to exchange public keys through a trusted third party, giving you a strong assurance that the public key really belongs to the entity with which you are communicating. Instead of sending your public key directly, you ask the trusted third party to incorporate it into a digital certificate. The trusted third party that issues digital certificates is called a Certificate Authority (CA), as described in "Certificate Authorities" on page 14.

What is in a digital certificate

Digital certificates contain specific pieces of information, as determined by the X.509 standard.

Digital certificates used by IBM MQ comply with the X.509 standard, which specifies the information that is required and the format for sending it. X.509 is the Authentication framework part of the X.500 series of standards.

Digital certificates contain at least the following information about the entity being certified:

- The owner's public key
- The owner's Distinguished Name
- The Distinguished Name of the CA that issued the certificate
- The date from which the certificate is valid
- The expiry date of the certificate
- The version number of the certificate data format as defined in X.509. The current version of the X.509 standard is Version 3, and most certificates conform to that version.
- A serial number. This is a unique identifier assigned by the CA which issued the certificate. The serial number is unique within the CA which issued the certificate: no two certificates signed by the same CA certificate have the same serial number.

An X.509 Version 2 certificate also contains an Issuer Identifier and a Subject Identifier, and an X.509 Version 3 certificate can contain a number of extensions. Some certificate extensions, such as the Basic Constraint extension, are *standard*, but others are implementation-specific. An extension can be *critical*, in which case a system must be able to recognize the field; if it does not recognize the field, it must reject the certificate. If an extension is not critical, the system can ignore it if it does not recognize it.

The digital signature in a personal certificate is generated using the private key of the CA which signed that certificate. Anyone who needs to verify the personal certificate can use the CA's public key to do so. The CA's certificate contains its public key.

Digital certificates do not contain your private key. You must keep your private key secret.

Requirements for personal certificates

IBM MQ supports digital certificates that comply with the X.509 standard. It requires the client authentication option.

Because IBM MQ is a peer to peer system, it is viewed as client authentication in SSL/TLS terminology. Therefore, any personal certificate used for SSL/TLS authentication needs to allow a key usage of client authentication. Not all server certificates have this option enabled, so the certificate provider might need to enable client authentication on the root CA for the secure certificate.

In addition to the standards which specify the data format for a digital certificate, there are also standards for determining whether a certificate is valid. These standards have been updated over time in order to prevent certain types of security breach. For example, older X.509 version 1 and 2 certificates did not indicate whether the certificate could be legitimately used to sign other certificates. It was therefore possible for a malicious user to obtain a personal certificate from a legitimate source and create new certificates designed to impersonate other users.

When using X.509 version 3 certificates, the BasicConstraints and KeyUsage certificate extensions are used to specify which certificates can legitimately sign other certificates. The IETF RFC 5280 standard specifies a series of certificate validation rules which compliant application software must implement in order to prevent impersonation attacks. A set of certificate rules is known as a certificate validation policy.

For more information about certificate validation policies in IBM MQ, see “Certificate validation policies in IBM MQ” on page 45.

Certificate Authorities

A Certificate Authority (CA) is a trusted third party that issues digital certificates to provide you with an assurance that the public key of an entity truly belongs to that entity.

The roles of a CA are:

- On receiving a request for a digital certificate, to verify the identity of the requestor before building, signing and returning the personal certificate
- To provide the CA's own public key in its CA certificate
- To publish lists of certificates that are no longer trusted in a Certificate Revocation List (CRL). For more information, see "Working with revoked certificates" on page 332
- To provide access to certificate revocation status by operating an OCSP responder server

Distinguished Names

The Distinguished Name (DN) uniquely identifies an entity in an X.509 certificate.

Attention: Only the attributes in the following table can be used in an SSLPEER filter. Certificate DNs can contain other attributes, but filtering is not allowed on these attributes.

Table 1. Attribute types found in the DN that can be used in an SSLPEER filter

Attribute type	**Description**
SERIALNUMBER	Certificate serial number
MAIL	Email address
Deprecated E	Email address (Deprecated in preference to MAIL)
UID or USERID	User identifier
CN	Common Name
T	Title
OU	Organizational Unit name
DC	Domain component
O	Organization name
STREET	Street / First line of address
L	Locality name
ST (or SP or S)	State or Province name
PC	Postal code / zip code
C	Country
UNSTRUCTUREDNAME	Host name
UNSTRUCTUREDADDRESS	IP address
DNQ	Distinguished name qualifier

The X.509 standard defines other attributes that do not typically form part of the DN but can provide optional extensions to the digital certificate.

The X.509 standard provides for a DN to be specified in a string format. For example:

```
CN=John Smith, OU=Test, O=IBM, C=GB
```

The Common Name (CN) can describe an individual user or any other entity, for example a web server.

The DN can contain multiple OU and DC attributes. Only one instance of each of the other attributes is permitted. The order of the OU entries is significant: the order specifies a hierarchy of Organizational Unit names, with the highest-level unit first. The order of the DC entries is also significant.

IBM MQ tolerates certain malformed DNs. For more information, see IBM MQ rules for SSLPEER values.

Related concepts

"What is in a digital certificate" on page 14
Digital certificates contain specific pieces of information, as determined by the X.509 standard.

Obtaining personal certificates from a certificate authority

You can obtain a certificate from a trusted external certificate authority (CA).

You obtain a digital certificate by sending information to a CA, in the form of a certificate request. The X.509 standard defines a format for this information, but some CAs have their own format. Certificate requests are typically generated by the certificate management tool your system uses; for example:

- ALW The **runmqakm** and V 9.4.0 V 9.4.0 **runmqktool** commands on AIX, Linux, and Windows.
- z/OS RACF® on z/OS.

The information contains your Distinguished Name and your public key. When your certificate management tool generates your certificate request, it also generates your private key, which you must keep secure. Never distribute your private key.

When the CA receives your request, the authority verifies your identity before building the certificate and returning it to you as a personal certificate.

Figure 3 on page 16 illustrates the process of obtaining a digital certificate from a CA.

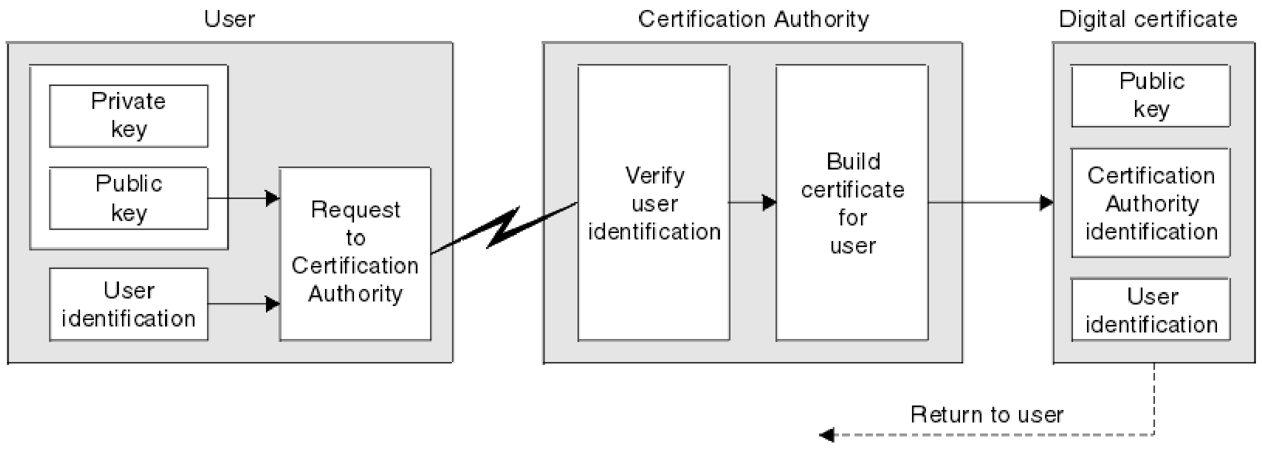


Figure 3. Obtaining a digital certificate

In the diagram:

- User identification includes your Subject Distinguished Name.
- Certification Authority identification includes the Distinguished Name of the CA that is issuing the certificate.

Digital certificates contain additional fields other than those shown in the diagram. For more information about the other fields in a digital certificate, see "What is in a digital certificate" on page 14.

How certificate chains work

When you receive the certificate for another entity, you might need to use a *certificate chain* to obtain the *root CA* certificate.

The certificate chain, also known as the *certification path*, is a list of certificates used to authenticate an entity. The chain, or path, begins with the certificate of that entity, and each certificate in the chain is signed by the entity identified by the next certificate in the chain. The chain terminates with a root CA certificate. The root CA certificate is always signed by the certificate authority (CA) itself. The signatures of all certificates in the chain must be verified until the root CA certificate is reached.

Figure 4 on page 17 illustrates a certification path from the certificate owner to the root CA, where the chain of trust begins.

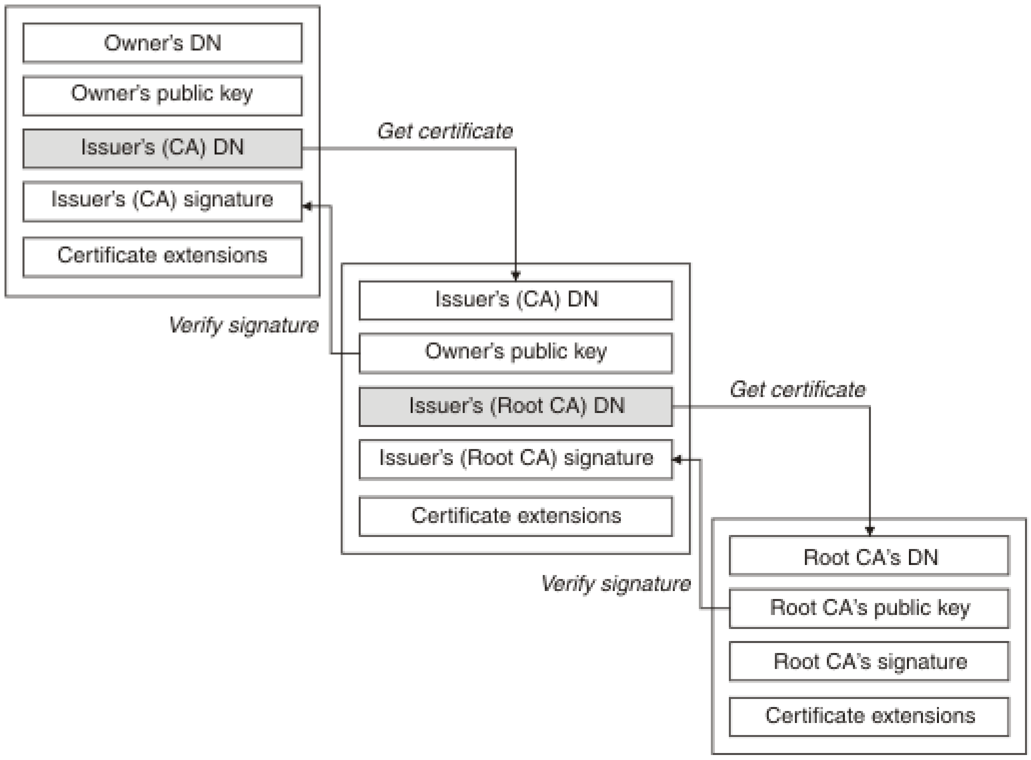


Figure 4. Chain of trust

Each certificate can contain one or more extensions. A certificate belonging to a CA typically contains a BasicConstraints extension with the isCA flag set to indicate that it is allowed to sign other certificates.

When certificates are no longer valid

Digital certificates can expire or be revoked.

Digital certificates are issued for a fixed period and are not valid after their expiry date.

Certificates can be revoked for various reasons, including:

- The owner has moved to a different organization.
- The private key is no longer secret.

IBM MQ can check whether a certificate is revoked by sending a request to an Online Certificate Status Protocol (OCSP) responder (on AIX, Linux, and Windows only). Alternatively, they can access a Certificate Revocation List (CRL) on an LDAP server. The OCSP revocation and CRL information is published by a Certificate Authority. For more information, see "Working with revoked certificates" on page 332.

Public Key Infrastructure (PKI)

A Public Key Infrastructure (PKI) is a system of facilities, policies, and services that supports the use of public key cryptography for authenticating the parties involved in a transaction.

There is no single standard that defines the components of a Public Key Infrastructure, but a PKI typically comprises certificate authorities (CAs) and Registration Authorities (RAs). CAs provide the following services::

- Issuing digital certificates
- Validating digital certificates
- Revoking digital certificates

- Distributing public keys

The X.509 standards provide the basis for the industry standard Public Key Infrastructure.

Refer to "Digital certificates" on page 13 for more information about digital certificates and certificate authorities (CAs). RAs verify the information provided when digital certificates are requested. If the RA verifies that information, the CA can issue a digital certificate to the requester.

A PKI might also provide tools for managing digital certificates and public keys. A PKI is sometimes described as a *trust hierarchy* for managing digital certificates, but most definitions include additional services. Some definitions include encryption and digital signature services, but these services are not essential to the operation of a PKI.

Cryptographic security protocols: TLS

Cryptographic protocols provide secure connections, enabling two parties to communicate with privacy and data integrity. The Transport Layer Security (TLS) protocol evolved from that of the Secure Sockets Layer (SSL). IBM MQ supports TLS.

The primary goals of both protocols is to provide confidentiality, (sometimes referred to as *privacy*), data integrity, identification, and authentication using digital certificates.

Although the two protocols are similar, the differences are sufficiently significant that SSL 3.0 and the various versions of TLS do not interoperate.

Related concepts

"TLS security protocols in IBM MQ " on page 24
IBM MQ supports the Transport Layer Security (TLS) protocol to provide link level security for message channels and MQI channels.

Transport Layer Security (TLS) concepts

The TLS protocol enables two parties to identify and authenticate each other and communicate with confidentiality and data integrity. The TLS protocol evolved from the Netscape SSL 3.0 protocol but TLS and SSL do not interoperate.

The TLS protocol provides communications security over the internet, and allow client/server applications to communicate in a way that is confidential and reliable. The protocols have two layers: a Record Protocol and a Handshake Protocol, and these are layered above a transport protocol such as TCP/IP. They both use asymmetric and symmetric cryptography techniques.

A TLS connection is initiated by an application, which becomes the TLS client. The application which receives the connection becomes the TLS server. Every new session begins with a handshake, as defined by the TLS protocols.

A full list of CipherSpecs supported by IBM MQ is provided at "Enabling CipherSpecs" on page 411.

For more information about the SSL protocol, see the information provided at https://developer.mozilla.org/docs/Mozilla/Projects/NSS. For more information about the TLS protocol, see the information provided by the TLS Working Group on the website of the Internet Engineering Task Force at https://www.ietf.org

An overview of the SSL/TLS handshake

The SSL/TLS handshake enables the TLS client and server to establish the secret keys with which they communicate.

This section provides a summary of the steps that enable the TLS client and server to communicate with each other.

- Agree on the version of the protocol to use.
- Select cryptographic algorithms.
- Authenticate each other by exchanging and validating digital certificates.

- Use asymmetric encryption techniques to generate a shared secret key, which avoids the key distribution problem. TLS then uses the shared key for the symmetric encryption of messages, which is faster than asymmetric encryption.

For more information about cryptographic algorithms and digital certificates, refer to the related information.

In overview, the steps involved in the TLS handshake are as follows:

1. The TLS client sends a "client hello" message that lists cryptographic information such as the TLS version and, in the client's order of preference, the CipherSuites supported by the client. The message also contains a random byte string that is used in subsequent computations. The protocol allows for the "client hello" to include the data compression methods supported by the client.
2. The TLS server responds with a "server hello" message that contains the CipherSuite chosen by the server from the list provided by the client, the session ID, and another random byte string. The server also sends its digital certificate. If the server requires a digital certificate for client authentication, the server sends a "client certificate request" that includes a list of the types of certificates supported and the Distinguished Names of acceptable Certification Authorities (CAs).
3. The TLS client verifies the server's digital certificate. For more information, see "How TLS provides identification, authentication, confidentiality, and integrity" on page 20.
4. The TLS client sends the random byte string that enables both the client and the server to compute the secret key to be used for encrypting subsequent message data. The random byte string itself is encrypted with the server's public key.
5. If the TLS server sent a "client certificate request", the client sends a random byte string encrypted with the client's private key, together with the client's digital certificate, or a "no digital certificate alert". This alert is only a warning, but with some implementations the handshake fails if client authentication is mandatory.
6. The TLS server verifies the client's certificate. For more information, see "How TLS provides identification, authentication, confidentiality, and integrity" on page 20.
7. The TLS client sends the server a "finished" message, which is encrypted with the secret key, indicating that the client part of the handshake is complete.
8. The TLS server sends the client a "finished" message, which is encrypted with the secret key, indicating that the server part of the handshake is complete.
9. For the duration of the TLS session, the server and client can now exchange messages that are symmetrically encrypted with the shared secret key.

Figure 5 on page 20 illustrates the TLS handshake.

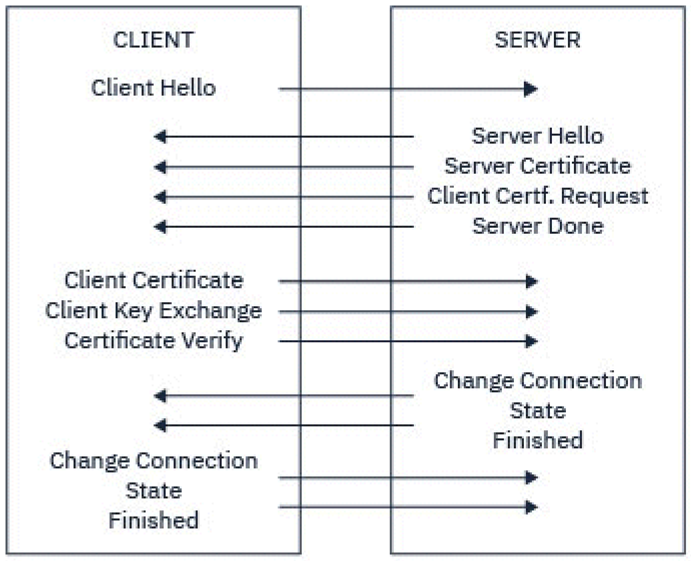


Figure 5. Overview of the TLS handshake

How TLS provides identification, authentication, confidentiality, and integrity

During both client and server authentication there is a step that requires data to be encrypted with one of the keys in an asymmetric key pair and decrypted with the other key of the pair. A message digest is used to provide integrity.

For an overview of the steps involved in the TLS handshake, see "An overview of the SSL/TLS handshake" on page 18.

How TLS provides authentication

For server authentication, the client uses the server's public key to encrypt the data that is used to compute the secret key. The server can generate the secret key only if it can decrypt that data with the correct private key. The random byte string itself is encrypted with the server's public key (step "4" on page 19 in the overview).

For client authentication, the server uses the public key in the client certificate to decrypt the data the client sends during step "5" on page 19 of the handshake. The exchange of finished messages that are encrypted with the secret key (steps "7" on page 19 and "8" on page 19 in the overview) confirms that authentication is complete.

If any of the authentication steps fail, the handshake fails and the session terminates.

The exchange of digital certificates during the TLS handshake is part of the authentication process. For more information about how certificates provide protection against impersonation, refer to the related information. The certificates required are as follows, where CA X issues the certificate to the TLS client, and CA Y issues the certificate to the TLS server:

For server authentication only, the TLS server needs:

- The personal certificate issued to the server by CA Y
- The server's private key

and the TLS client needs:

- The CA certificate for CA Y

If the TLS server requires client authentication, the server verifies the client's identity by verifying the client's digital certificate with the public key for the CA that issued the personal certificate to the client, in this case CA X. For both server and client authentication, the server needs:

- The personal certificate issued to the server by CA Y

- The server's private key
- The CA certificate for CA X

and the client needs:

- The personal certificate issued to the client by CA X
- The client's private key
- The CA certificate for CA Y

Both the TLS server and client might need other CA certificates to form a certificate chain to the root CA certificate. For more information about certificate chains, refer to the related information.

What happens during certificate verification

As noted in steps “3” on page 19 and “6” on page 19 of the overview, the TLS client verifies the server's certificate, and the TLS server verifies the client's certificate. There are four aspects to this verification:

1. The digital signature is checked (see “Digital signatures in SSL/TLS” on page 22).
2. The certificate chain is checked; you should have intermediate CA certificates (see “How certificate chains work” on page 16).
3. The expiry and activation dates and the validity period are checked.
4. The revocation status of the certificate is checked (see “Working with revoked certificates” on page 332).

Secret key reset

During a TLS handshake a *secret key* is generated to encrypt data between the TLS client and server. The secret key is used in a mathematical formula that is applied to the data to transform plaintext into unreadable ciphertext, and ciphertext into plaintext.

The secret key is generated from the random text sent as part of the handshake and is used to encrypt plaintext into ciphertext. The secret key is also used in the MAC (Message Authentication Code) algorithm, which is used to determine whether a message has been altered. See “Message digests and digital signatures” on page 13 for more information.

If the secret key is discovered, the plaintext of a message could be deciphered from the ciphertext, or the message digest could be calculated, allowing messages to be altered without detection. Even for a complex algorithm, the plaintext can eventually be discovered by applying every possible mathematical transformation to the ciphertext. To minimize the amount of data that can be deciphered or altered if the secret key is broken, the secret key can be renegotiated periodically. When the secret key has been renegotiated, the previous secret key can no longer be used to decrypt data encrypted with the new secret key.

How TLS provides confidentiality

TLS uses a combination of symmetric and asymmetric encryption to ensure message privacy. During the TLS handshake, the TLS client and server agree an encryption algorithm and a shared secret key to be used for one session only. All messages transmitted between the TLS client and server are encrypted using that algorithm and key, ensuring that the message remains private even if it is intercepted. Because TLS uses asymmetric encryption when transporting the shared secret key, there is no key distribution problem. For more information about encryption techniques, refer to “Cryptography” on page 11.

How TLS provides integrity

TLS provides data integrity by calculating a message digest. For more information, refer to “Data integrity of messages” on page 465.

Use of TLS does ensure data integrity, provided that the CipherSpec in your channel definition uses a hash algorithm as described in the table in “Enabling CipherSpecs” on page 411.

In particular, if data integrity is a concern, you should avoid choosing a CipherSpec whose hash algorithm is listed as "None". Use of MD5 is also strongly discouraged as this is now very old and no longer secure for most practical purposes.

CipherSpecs and CipherSuites

Cryptographic security protocols must agree on the algorithms used by a secure connection. CipherSpecs and CipherSuites define specific combinations of algorithms.

A CipherSpec identifies a combination of encryption algorithm and Message Authentication Code (MAC) algorithm. Both ends of a TLS connection must agree on the same CipherSpec to be able to communicate.

IBM MQ supports TLS1.3 and TLS1.2 protocols and CipherSpecs. However, you can enable deprecated CipherSpecs, if you need to do so.

See “Enabling CipherSpecs” on page 411 for information on:

- CipherSpecs supported by IBM MQ
- How you enable deprecated SSL 3.0 and TLS 1.0 CipherSpecs

Important: When dealing with IBM MQ channels, you use a CipherSpec. When dealing with Java channels, JMS channels, or MQTT channels you specify a CipherSuite.

For more information about CipherSpecs, see “Enabling CipherSpecs” on page 411.

A CipherSuite is a suite of cryptographic algorithms used by a TLS connection. A suite comprises three distinct algorithms:

- The key exchange and authentication algorithm, used during the handshake
- The encryption algorithm, used to encipher the data
- The MAC (Message Authentication Code) algorithm, used to generate the message digest

There are several options for each component of the suite, but only certain combinations are valid when specified for a TLS connection. The name of a valid CipherSuite defines the combination of algorithms used. For example, the CipherSuite TLS_RSA_WITH_AES_128_CBC_SHA specifies:

- The RSA key exchange and authentication algorithm
- The AES encryption algorithm, using a 128-bit key and cipher block chaining (CBC) mode
- The SHA-1 Message Authentication Code (MAC)

Digital signatures in SSL/TLS

A digital signature is formed by encrypting a representation of a message. The encryption uses the private key of the signatory and, for efficiency, usually operates on a message digest rather than the message itself.

Digital signatures vary with the data being signed, unlike handwritten signatures, which do not depend on the content of the document being signed. If two different messages are signed digitally by the same entity, the two signatures differ, but both signatures can be verified with the same public key, that is, the public key of the entity that signed the messages.

The steps of the digital signature process are as follows:

1. The sender computes a message digest and then encrypts the digest using the sender's private key, forming the digital signature.
2. The sender transmits the digital signature with the message.
3. The receiver decrypts the digital signature using the sender's public key, regenerating the sender's message digest.
4. The receiver computes a message digest from the message data received and verifies that the two digests are the same.

Figure 6 on page 23 illustrates this process.

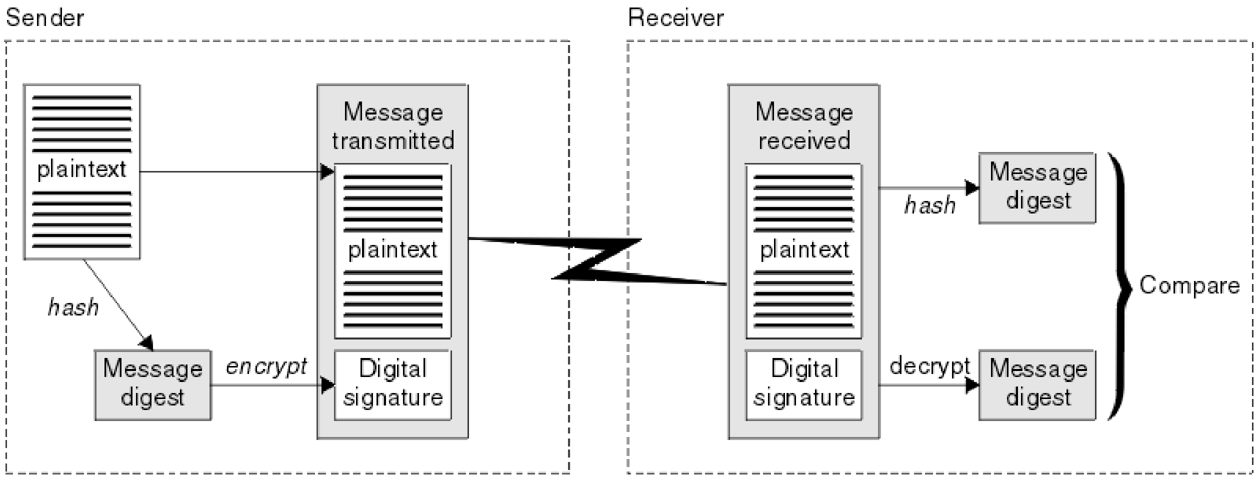


Figure 6. The digital signature process

If the digital signature is verified, the receiver knows that:

- The message has not been modified during transmission.
- The message was sent by the entity that claims to have sent it.

Digital signatures are part of integrity and authentication services. Digital signatures also provide proof of origin. Only the sender knows the private key, which provides strong evidence that the sender is the originator of the message.

Note: You can also encrypt the message itself, which protects the confidentiality of the information in the message.

Federal Information Processing Standards

The US government produces technical advice on IT systems and security, including data encryption. The National Institute for Standards and Technology (NIST) is an important body concerned with IT systems and security. NIST produces recommendations and standards, including the Federal Information Processing Standards (FIPS).

A significant one of these standards is FIPS 140-2, which requires the use of strong cryptographic algorithms. FIPS 140-2 also specifies requirements for hashing algorithms to be used to protect packets against modification in transit.

Note: On AIX, Linux, and Windows, IBM MQ provides FIPS 140-2 compliance through the IBM Crypto for C (ICC) cryptographic module. The certificate for this module has been moved to the Historical status. Customers should view the IBM Crypto for C (ICC) certificate and be aware of any advice provided by NIST. A replacement FIPS 140-3 module is currently in progress and its status can be viewed by searching for it in the NIST CMVP modules in process list.

The IBM MQ Operator 3.2.0 and queue manager container image 9.4.0.0 onwards are based on UBI 9. FIPS 140-3 compliance is currently pending and its status can be viewed by searching for "Red Hat Enterprise Linux 9 - OpenSSL FIPS Provider" in the NIST CMVP modules in process list.

IBM MQ provides FIPS 140-2 support when it has been configured to do so.

Over time, analysts develop attacks against existing encryption and hashing algorithms. New algorithms are adopted to resist those attacks. FIPS 140-2 is periodically updated to take account of these changes.

Related concepts

“National Security Agency (NSA) Suite B Cryptography” on page 24

The government of the Unites States of America produces technical advice on IT systems and security, including data encryption. The US National Security Agency (NSA) recommends a set of interoperable cryptographic algorithms in its Suite B standard.

National Security Agency (NSA) Suite B Cryptography

The government of the Unites States of America produces technical advice on IT systems and security, including data encryption. The US National Security Agency (NSA) recommends a set of interoperable cryptographic algorithms in its Suite B standard.

The Suite B standard specifies a mode of operation in which only a specific set of secure cryptographic algorithms are used. The Suite B standard specifies:

- The encryption algorithm (AES)
- The key exchange algorithm (Elliptic Curve Diffie-Hellman, also known as ECDH)
- The digital signature algorithm (Elliptic Curve Digital Signature Algorithm, also known as ECDSA)
- The hashing algorithms (SHA-256 or SHA-384)

Additionally, the IETF RFC 6460 standard specifies Suite B compliant profiles which define the detailed application configuration and behavior necessary to comply with the Suite B standard. It defines two profiles:

1. A Suite B compliant profile for use with TLS 1.2. When configured for Suite B compliant operation, only the restricted set of cryptographic algorithms listed are used.
2. A transitional profile for use with TLS 1.0 or TLS 1.1. This profile enables interoperability with non-Suite B compliant servers. When configured for Suite B transitional operation, additional encryption and hashing algorithms may be used.

The Suite B standard is conceptually similar to FIPS 140-2, because it restricts the set of enabled cryptographic algorithms in order to provide an assured level of security.

On AIX, Linux, and Windows systems, IBM MQ, can be configured to conform to the Suite B compliant TLS 1.2 profile, but does not support the Suite B transitional profile. For further information, see "NSA Suite B Cryptography in IBM MQ" on page 42.

Related reference

"Federal Information Processing Standards" on page 23
The US government produces technical advice on IT systems and security, including data encryption. The National Institute for Standards and Technology (NIST) is an important body concerned with IT systems and security. NIST produces recommendations and standards, including the Federal Information Processing Standards (FIPS).

IBM MQ security mechanisms

This collection of topics describes specific mechanisms in IBM MQ that implement the various security concepts.

TLS security protocols in IBM MQ

IBM MQ supports the Transport Layer Security (TLS) protocol to provide link level security for message channels and MQI channels.

Message channels and MQI channels can use the TLS protocol to provide link level security. A caller MCA is a TLS client and a responder MCA is a TLS server.

IBM MQ supports Versions 1.2 and 1.3 of the TLS protocol. Earlier versions of TLS, as well as SSL, are not enabled by default, but can be if needed. You can specify the cryptographic algorithms that are used by the TLS protocol by supplying a CipherSpec as part of the channel definition.

See "Enabling CipherSpecs" on page 411 for a list of the CipherSpecs supported by IBM MQ and "Deprecated CipherSpecs" on page 425 for those that are deprecated.

You can use the SECPROT and SSLCIPH parameters to display the security protocol and CipherSpec in use on a channel.

At each end of a message channel, and at the server end of an MQI channel, the MCA acts on behalf of the queue manager to which it is connected. During the TLS handshake, the MCA sends the digital certificate of the queue manager to its partner MCA at the other end of the channel. The IBM MQ code at the client end of an MQI channel acts on behalf of the user of the IBM MQ client application. During the TLS handshake, the IBM MQ code sends the user's digital certificate to the MCA at the server end of the MQI channel.

Queue managers and IBM MQ client users are not required to have personal digital certificates associated with them when they are acting as TLS clients, unless SSLCAUTH(REQUIRED) is specified at the server side of the channel.

Digital certificates are stored in a *key repository*. The queue manager attribute **SSLKeyRepository** specifies the location of the key repository that holds the queue manager's digital certificate. On an IBM MQ client system, the MQSSLKEYR environment variable specifies the location of the key repository that holds the user's digital certificate. Alternatively, an IBM MQ client application can specify its location in the **KeyRepository** field of the TLS configuration options structure, MQSCO, on an MQCONNX call. See the related topics for more information about key repositories and how to specify where they are located.

Support for TLS

IBM MQ provides support for TLS 1.2 and TLS 1.3 on all platforms. For more information about the TLS protocol, refer to the information in the subtopics.

Java and JMS clients
: These clients use the JVM to provide TLS support.

AIX, Linux, and Windows
: TLS support is installed with IBM MQ.

IBM i
: TLS support is integral to the IBM i operating system.

z/OS
: TLS support is integral to the z/OS operating system. The TLS support on z/OS is known as *System SSL*.

For information about any prerequisites for IBM MQ TLS support, see System Requirements for IBM MQ.

Related concepts

"Cryptographic security protocols: TLS" on page 18
Cryptographic protocols provide secure connections, enabling two parties to communicate with privacy and data integrity. The Transport Layer Security (TLS) protocol evolved from that of the Secure Sockets Layer (SSL). IBM MQ supports TLS.

The SSL/TLS key repository

A mutually authenticated TLS connection requires a key repository at each end of the connection. The key repository includes digital certificates and private keys.

This information uses the general term *key repository* to describe the store for digital certificates and their associated private keys. The key repository is referred to by different names on different platforms and environments that support TLS:

- IBM i On IBM i: *certificate store*
- On Java and JMS: *keystore* and *truststore*
- ALW On AIX, Linux, and Windows: *key database file*
- z/OS On z/OS: *keyring*

For more information, see "Digital certificates" on page 13 and "Transport Layer Security (TLS) concepts" on page 18.

A mutually authenticated TLS connection requires a key repository at each end of the connection. The key repository can contain the following certificates and requests:

- A number of CA certificates from various Certification Authorities that allow the queue manager or client to verify certificates that it receives from its partner at the remote end of the connection. Individual certificates might be in a certificate chain.
- One or more personal certificates received from a Certification Authority. You associate a separate personal certificate with each queue manager or IBM MQ MQI client. Personal certificates are essential on a TLS client if mutual authentication is required. If mutual authentication is not required, personal certificates are not needed on the client. The key repository might also contain the private key corresponding to each personal certificate.
- Certificate requests which are waiting to be signed by a trusted CA certificate.

For more information about protecting your key repository, see "Protecting IBM MQ key repositories" on page 27.

The location of the key repository depends on the platform you are using:

IBM i **IBM i**

The key repository is a certificate store. The default system certificate store is located at `/QIBM/UserData/ICSS/Cert/Server/Default` in the integrated file system (IFS). IBM MQ stores the password for the certificate store in a *password stash file*. For example, the stash file for queue manager QM1 is `/QIBM/UserData/mqm/qmgrs/QM1/ssl/Stash.sth`.

Alternatively, you can specify that the IBM i system certificate store is to be used instead. To do this change the value of the queue manager **SSLKEYR** attribute to *SYSTEM. This value indicates that the queue manager must use the system certificate store, and the queue manager is registered for use as an application with Digital Certificate Manager (DCM).

The certificate store also contains the private key for the queue manager.

ALW **AIX, Linux, and Windows systems**

The key repository is a key database file. For example, on AIX and Linux, the default key database file for queue manager QM1 is `/var/mqm/qmgrs/QM1/ssl/key.kdb`. If IBM MQ is installed in the default location, the equivalent path on Windows is `C:\ProgramData\IBM\MQ\Qmgrs\QM1\ssl\key.kdb`.

To access a key database file IBM MQ must be supplied the password for the key database. This can be done either directly or through a password stash file. If a password stash file is used, it must be in the same directory and have the same file stem as the key database, and must end with the suffix `.sth`, for example, `/var/mqm/qmgrs/QM1/ssl/key.sth`.

Note: PKCS #11 cryptographic hardware cards can contain the certificates and keys that are otherwise held in a key database file. When certificates and keys are held on PKCS #11 cards, IBM MQ still requires access to both a key database file and a password stash file.

On AIX, Linux, and Windows systems, the key database also contains the private key for the personal certificate associated with the queue manager or IBM MQ MQI client.

z/OS **z/OS**

Certificates are held in a keyring in z/OS.

Other external security managers (ESMs) also use keyrings for storing certificates.

Private keys are managed by RACF.

Protecting IBM MQ key repositories
The key repository for IBM MQ is a file. Ensure that only the intended user can access the key repository file. This prevents an intruder or other unauthorized user copying the key repository file to another system, and then setting up an identical user ID on that system to impersonate the intended user.

The permissions on the files depend on the user's umask and which tool is used. On Windows, IBM MQ accounts require permission `BypassTraverseChecking` which means the permissions of the folders in the file path have no effect.

Check the file permissions of key repository files and make sure that the files and containing folder are not world readable, preferably not even group readable.

Making the keystore read-only is good practice, on whichever system you use, with only the administrator being permitted to enable write operations in order to perform maintenance.

In practice, you must protect all the keystores, whatever the location and whether they are password protected or not; protect the key repositories.

Digital certificate labels, understanding the requirements
When setting up TLS to use digital certificates, there might be specific label requirements that you must follow, depending on the platform used and the method you use to connect.

What is the certificate label?

A certificate label is a unique identifier representing a digital certificate stored in a key repository, and provides a convenient human-readable name with which to refer to a particular certificate when performing key management functions. You assign the certificate label when adding a certificate to a key repository for the first time.

The certificate label is separate from the certificate's **`Subject Distinguished Name`** or **`Subject Common Name`** fields. Note that **`Subject Distinguished Name`** and **`Subject Common Name`** are fields within the certificate itself. These are defined when the certificate is created and cannot be changed. If necessary, however, you can change the label associated with a digital certificate.

Certificate label syntax

A certificate label can contain letters, numbers, and punctuation with the following conditions:

- Multi The certificate label can contain up to 64 characters.
- z/OS The certificate label can contain up to 32 characters.
- The certificate label can contain spaces.
- Labels are case sensitive.
- On systems that use EBCDIC katakana, you cannot use lowercase characters.

Additional requirements for certificate label values are specified in the following sections.

How is the certificate label used?

IBM MQ uses certificate labels to locate a personal certificate that is sent during the TLS handshake. This eliminates ambiguity when more than one personal certificate exists in the key repository.

You can set the certificate label to a value of your choice. If you do not set a value, a default label is used that follows a naming convention depending on the platform that you are using. For details, see the sections that follow, about particular platforms.

Notes:

1. You cannot set the certificate label yourself on Java or JMS systems.

2. Auto-defined channels created by a channel automatic definition (CHAD) exit cannot set the certificate label, because the TLS handshake has occurred by the time the channel is created. Setting the certificate label in a CHAD exit for inbound channels has no effect.

In this context, a TLS client refers to the connection partner initiating the handshake, which might be an IBM MQ client or another queue manager.

During the TLS handshake, the TLS client always obtains and validates a digital certificate from the server. With the IBM MQ implementation, the TLS server always requests a certificate from the client and the client always provides a certificate to the server if one is found. If the client is unable to locate a personal certificate, the client sends a `no certificate` response to the server.

The TLS server always validates the client certificate if one is sent. If the client does not send a certificate, authentication fails if the end of the channel that is acting as the TLS server is defined with either the **SSLCAUTH** parameter set to *REQUIRED* or an **SSLPEER** parameter value set.

Note that inbound channels (including receiver, requester, cluster-receiver, unqualified server, and server-connection channels) only send the configured certificate if the IBM MQ version of the remote peer fully supports certificate label configuration, and the channel is using a TLS CipherSpec.

An unqualified server channel is one that does not have the CONNAME field set.

In all other cases, the queue manager **CERTLABL** parameter determines the certificate sent. In particular, the following only ever receive the certificate configured by the **CERTLABL** parameter of the queue manager, regardless of the channel-specific label setting:

- Java and JMS clients supporting Server Name Indication (SNI), that is, certificates on a channel by channel basis.
- Versions of IBM MQ prior to IBM MQ 8.0.
- Managed .NET clients

Additionally, the certificate used by a channel must be appropriate for the channel CipherSpec - see "Digital certificates and CipherSpec compatibility in IBM MQ" on page 47 for further information.

IBM MQ 8.0 and later supports the use of multiple certificates on the same queue manager, using a per-channel certificate label, specified using the **CERTLABL** attribute on the channel definition. Inbound channels to the queue manager (for example, server connection or receiver) rely on detecting the channel name using TLS Server Name Indication (SNI), in order to present the correct certificate from the queue manager. For more information about using multiple certificates on a queue manager, see "How IBM MQ provides multiple certificates capability " on page 30.

If a channel connects to the destination queue manager through IBM MQ Internet Pass-Thru (MQIPT), and the MQIPT route has both **`SSLServer`** and **`SSLClient`** set, there are two separate TLS sessions between the endpoints. MQIPT can be configured to allow multiple certificates to be used by the destination queue manager by either setting the SNI to the channel name, or by passing through the SNI received on the inbound connection to the route. For more information about multiple certificate support and MQIPT, see IBM MQ multiple certificate support with MQIPT.

For more information about connecting a queue manager using one-way authentication, that is, when the TLS client does not send a certificate, see Connecting two queue managers using one-way authentication.

Multiplatforms systems

Multi

On Multiplatforms, the TLS server sends a certificate to the client.

For queue managers and clients respectively, the following sources are searched in sequence for a non-empty value. The first non-empty value determines the certificate label. The certificate label must exist in the key repository. If no matching certificate in the correct case and format is found that matches a label, an error occurs and the TLS handshake fails.

Queue managers

1. Channel certificate label attribute **CERTLABL**.
2. Queue manager certificate label attribute **CERTLABL**.
3. A default, which is in the format: `ibmwebspheremq` with the name of the queue manager appended, all in lowercase. For example, for a queue manager named `QM1`, the default certificate label is `ibmwebspheremqqm1`.

IBM MQ clients

1. Certificate label attribute **CERTLABL** in the CLNTCONN channel definition.
2. MQSCO structure **CertificateLabel** attribute.
3. Environment variable **MQCERTLABL**.
4. Client `.ini` file (in its SSL section) **CertificateLabel** attribute
5. A default, which is in the format: `ibmwebspheremq` with the user ID that the client application is running as appended, all in lowercase. For example, for a user ID of USER1, the default certificate label is `ibmwebspheremquser1`.

z/OS systems

z/OS

IBM MQ Clients are not supported on z/OS. However, a z/OS queue manager can act in the role of a TLS client when initiating a connection, or a TLS server when accepting a connection request. Certificate label requirements for z/OS queue managers apply in both of these roles, and differ from the requirements on Multiplatforms.

For queue managers and clients respectively, the following sources are searched in sequence for a non-empty value. The first non-empty value determines the certificate label. The certificate label must exist in the key repository. If no matching certificate in the correct case and format is found that matches a label, an error occurs and the TLS handshake fails.

1. Channel certificate label attribute, **CERTLABL**.
2. If shared, the queue sharing group certificate label attribute, **CERTQSGL**.

   If not shared, the queue manager certificate label attribute, **CERTLABL**.
3. A default, which is in the format: `ibmWebSphereMQ` with the name of the queue manager or queue sharing group appended. Note that this string is case-sensitive and must be written as shown. For example, for a queue manager named `QM1`, the default certificate label is `ibmWebSphereMQQM1`.
4. If there is not a certificate found with the format in option "3" on page 29, IBM MQ attempts to use the certificate marked as default in the key ring.

For information on how to display the key repository, see "Locating the key repository for a queue manager on z/OS" on page 303.

IBM MQ Java and IBM MQ JMS clients

IBM MQ Java and IBM MQ JMS clients use the facilities of their Java Secure Socket Extension (JSSE) provider to select a personal certificate during the TLS handshake and are not therefore subject to certificate label requirements.

The default behavior is that the JSSE client iterates through the certificates in the key repository, selecting the first acceptable personal certificate found. However, this behavior is only a default, and is dependent on the implementation of the JSSE provider.

In addition, the JSSE interface is highly customizable through configuration and direct access at runtime by the application. Consult the documentation supplied by your JSSE provider for specific details.

For troubleshooting, or to better understand the handshake performed by the IBM MQ Java client application in combination with your specific JSSE provider, you can enable debugging by setting `javax.net.debug=ssl` in the JVM environment.

You can set the variable within the application, through configuration, or by entering `-Djavax.net.debug=ssl` on the command line.

Linux *How IBM MQ provides multiple certificates capability*
Server Name Indication (SNI) is an extension to the TLS protocol that allows a client to indicate what service it requires. In IBM MQ terminology this equates to a channel.

The SNI extension is used by IBM MQ to allow multiple certificates to be specified across different channels using the CERTLABL parameter on the channel definition.

The SNI address used by IBM MQ is based upon the channel name that is being requested, followed by a suffix of `.chl.mq.ibm.com`.

IBM MQ channel names are mapped to be valid SNI names as follows:

- Upper case letters A to Z are folded to lower case
- Digits `0` to 9 are left unchanged
- All other characters, including lower case letters a to z, are converted into their two digit hexadecimal ASCII character code (in lower case), followed by a hyphen.
  - Lower case letters a to z map to hexadecimal `61-` to `7a-` respectively
  - percent (%) maps to hexadecimal `25-`
  - hyphen (`-`) maps to hexadecimal `2d-`
  - dot (`.`) maps to hexadecimal `2e-`
  - forward slash (/) maps to hexadecimal `2f-`
  - underscore (_) maps to hexadecimal `5f-`

On EBCDIC platforms, the channel name is converted to ASCII before this mapping is applied.

As an example, channel name `TO.QMGR1` maps to an SNI address of `to2e-qmgr1.chl.mq.ibm.com`.

By contrast, the lower case channel name `to.qmgr1` maps onto SNI address of `74-6f-2e-71-6d-67-72-1.chl.mq.ibm.com`.

Note: In environments where the generated SNI URL must conform to URL formatting specifications, for example when a client is connecting to a queue manager running in Red Hat® OpenShift® across a Red Hat OpenShift Route, the channel name must not end with a lower case letter.

The **`OutboundSNI`** property of the SSL stanza allows you to select whether the SNI should be set to the target IBM MQ channel name to the remote system when initiating a TLS connection, or to the hostname. For more information about the **`OutboundSNI`** property, see SSL stanza of the qm.ini file and SSL stanza of the client configuration file.

Multiple certificates requires that the SNI is set to the IBM MQ channel name. If a hostname, custom, or no SNI is used to connect to an IBM MQ channel with a certificate label configured, the connecting application is rejected with an MQRC_SSL_INITIALIZATION_ERROR, and an AMQ9673 message is printed in the remote queue manager error logs.

If a channel connects to the destination queue manager through IBM MQ Internet Pass-Thru (MQIPT), MQIPT must be configured to either set the SNI to the channel name, or to pass through the SNI received on the inbound connection to the route, to allow multiple certificates to be used by the destination queue manager. For more information about multiple certificate support and MQIPT, see IBM MQ multiple certificate support with MQIPT.

For more information on how this property is used, see Connecting to a queue manager deployed in a Red Hat OpenShift cluster.

Refreshing the queue manager's key repository
When you change the contents of a key repository, existing queue manager processes do not pick up the new contents until a REFRESH SECURITY TYPE(SSL) command is issued or the queue manager is restarted.

For more information about the REFRESH SECURITY TYPE(SSL) command, see REFRESH SECURITY.

If the queue manager creates a new channel process (using `amqrmppa` or **`runmqchl`**) after changing the contents of the keystore, the new process starts using the new certificates immediately, while existing processes continue to use their cached copy of the keystore. See "When changes to certificates or the key repository become effective on AIX, Linux, and Windows" on page 299 for more details.

Note, that multiple running channels could be using different versions of the key repository until you issue a REFRESH SECURITY TYPE(SSL) command.

You can also refresh a key repository using PCF commands or the IBM MQ Explorer. For more information, see the MQCMD_REFRESH_SECURITY command and the topic *Refreshing TLS Security* in the IBM MQ Explorer section of this product documentation.

Related concepts

"Refreshing a client's view of the SSL/TLS key repository contents and SSl/TLS settings" on page 31
To update the client application with the refreshed contents of the key repository, you must stop and restart the client application.

Refreshing a client's view of the SSL/TLS key repository contents and SSl/TLS settings
To update the client application with the refreshed contents of the key repository, you must stop and restart the client application.

You cannot refresh security on an IBM MQ client; there is no equivalent of the REFRESH SECURITY TYPE(SSL) command for clients (see REFRESH SECURITY) for more information.

You must stop and restart the application, whenever you change the security certificate, to update the client application with the refreshed contents of the key repository.

If restarting the channel refreshes the configurations, and if your application has reconnection logic, it is possible for you to refresh security at the client by issuing the STOP CHL STATUS(INACTIVE) command.

Related concepts

"Refreshing the queue manager's key repository" on page 31
When you change the contents of a key repository, existing queue manager processes do not pick up the new contents until a REFRESH SECURITY TYPE(SSL) command is issued or the queue manager is restarted.

MQCSP password protection

Authentication credentials that are specified in the MQCSP structure can be either protected by using the IBM MQ MQCSP password protection feature, or encrypted by using TLS encryption.

IBM MQ client applications can supply a user ID and password when they connect to a queue manager. V 9.4.0 From IBM MQ 9.4.0, applications can also supply an authentication token as an alternative method of authentication. These credentials are sent to the queue manager in an MQCSP structure.

If the channel is using TLS encryption, credentials in the MQCSP are encrypted according to the TLS cipher specification. If the channel is not using TLS encryption, IBM MQ can protect these credentials before they are sent over the network, to avoid sending credentials over a network in plain text. The IBM MQ feature that protects these credentials is called MQCSP password protection.

If MQCSP password protection is used, the following data in the MQCSP structure is protected:

- The password, if the `MQCSP.AuthenticationType` field is set to `MQCSP_AUTH_USER_ID_AND_PW`.
- V 9.4.0 The authentication token, if the `MQCSP.AuthenticationType` field is set to `MQCSP_AUTH_ID_TOKEN`.

Important: MQCSP password protection is useful for test and development purposes as using MQCSP password protection is simpler than setting up TLS encryption, but not as secure. For production purposes, use TLS encryption in preference to IBM MQ password protection, especially when the network between the client and queue manager is untrusted, as TLS encryption is more secure.

If you are concerned about what encryption is being used, and how much protection it offers, you need to use full TLS encryption. With TLS, the algorithms are publicly known, and you can select the appropriate one for your enterprise by using the **SSLCIPH** channel attribute.

For more information about the MQCSP structure, see MQCSP structure.

Credentials in the MQCSP structure are protected by using IBM MQ password protection if all the following conditions are met:

- Both ends of the connection are using IBM MQ 8.0, or later.
- The channel is not using TLS encryption. A channel is not using TLS encryption if the channel has a blank **SSLCIPH** attribute, or the **SSLCIPH** attribute is set to a cipher specification that does not provide encryption. Null ciphers, for example, NULL_SHA, do not provide encryption.
- The `MQCSP.AuthenticationType` field is set to `MQCSP_AUTH_USER_ID_AND_PWD` or `MQCSP_AUTH_ID_TOKEN`. For more information about the `MQCSP.AuthenticationType` field, see **AuthenticationType**.
- If the client is IBM MQ Explorer and user identification compatibility mode is not enabled. This mode is not the default mode that is used by IBM MQ Explorer to send a user ID and password. This condition is applicable only to IBM MQ Explorer.

If any of these conditions are not met, the credentials are not protected with MQCSP password protection. If the value of the **PasswordProtection** attribute prohibits credentials from being sent in plain text, and the channel is not using TLS encryption, the connection fails and an MQRC_PASSWORD_PROTECTION_ERROR (2594) reason code is returned.

The PasswordProtection configuration setting

The **PasswordProtection** attribute in the **Channels** stanza of the client and queue manager configuration files can prevent credentials from being sent in plain text.

Note: This attribute is only relevant for connections that do not use TLS encryption. Credentials are encrypted by using TLS instead of being protected with MQCSP password protection if the connection uses TLS encryption.

The attribute can be set to one of the following values. The default value is `compatible`.

compatible

Credentials are sent in plain text if either the queue manager or client is running a version of IBM MQ earlier than IBM MQ 8.0. That is, credentials can be sent over a network in plain text for compatibility with versions of IBM MQ that do not support MQCSP password protection.

Credentials are protected by MQCSP password protection if both the queue manager and the client are running a version of IBM MQ at IBM MQ 8.0 or later.

The connection fails before the credentials are sent if both the queue manager and the client are running a version of IBM MQ at IBM MQ 8.0 or later, and the `MQCSP.AuthenticationType` field is not set to `MQCSP_AUTH_USER_ID_AND_PW` or `MQCSP_AUTH_ID_TOKEN`.

always

Credentials must not be sent over a network unprotected.

Credentials are protected by MQCSP password protection if both the queue manager and the client are running a version of IBM MQ at IBM MQ 8.0 or later.

The connection fails before the credentials are sent in the following cases:

- The `MQCSP.AuthenticationType` field is not set to `MQCSP_AUTH_USER_ID_AND_PW` or `MQCSP_AUTH_ID_TOKEN`.
- Either the queue manager or the client is running a version of IBM MQ earlier than IBM MQ 8.0.

optional

Credentials are protected by MQCSP password protection if both the queue manager and the client are running a version of IBM MQ at IBM MQ 8.0 or later, and the `MQCSP.AuthenticationType` field is set to `MQCSP_AUTH_USER_ID_AND_PW` or `MQCSP_AUTH_ID_TOKEN`. Otherwise, credentials are sent in plain text.

warn

Any client is allowed to send plain text credentials. If plain text credentials are received, warning message AMQ9297W is written to the queue manager error logs.

This option can be specified only in the queue manager configuration file.

For Java and JMS clients, the behavior of the **`PasswordProtection`** attribute changes depending on whether the client uses compatibility mode or MQCSP mode:

- If Java and JMS clients are operating in compatibility mode, an MQCSP structure is not used to send the user ID and password when the client connects. Therefore, the behavior of the **`PasswordProtection`** attribute is the same as the behavior described for clients that are running a version of IBM MQ earlier than IBM MQ 8.0.
- If Java and JMS clients are operating in MQCSP mode, the behavior of the **`PasswordProtection`** attribute is the behavior as described.

For more information about connection authentication with Java and JMS clients, see "Connection authentication with the Java client" on page 82.

MQCSP password protection and MQIPT

V 9.4.0

If a client connects to a queue manager through IBM MQ Internet Pass-Thru (MQIPT), the MQIPT route might be configured to add or remove TLS encryption. That is, the MQIPT route might be configured with `SSLServer=true` and `SSLClient=false`, or `SSLServer=true` and `SSLClient=false`. In this situation, the client and the queue manager might fail to agree a password protection algorithm as one end of the channel is using TLS encryption, and the other is not. This causes the connection to fail with reason code MQRC_PASSWORD_PROTECTION_ERROR (2594).

From IBM MQ 9.4.0, MQIPT can add or remove protection for credentials in MQCSP structures, in order to maintain compatibility between the client and queue manager for MQIPT routes that add or remove TLS encryption. MQCSP password protection in MQIPT is configured by using the **`PasswordProtection`** route property.

The default value of the **`PasswordProtection`** property is `required`. This value means that MQIPT is able to add, but not remove, MQCSP password protection. Connections to a MQIPT route that adds TLS encryption might fail with reason code MQRC_PASSWORD_PROTECTION_ERROR (2594) with this value of **`PasswordProtection`**. To resolve this issue, set the value of the **`PasswordProtection`** property to `compatible` in the MQIPT route configuration.

For more information about the **`PasswordProtection`** property in MQIPT, see PasswordProtection.

Digital Certificate Manager (DCM)

Use the DCM to manage digital certificates and private keys on IBM i.

The Digital Certificate Manager (DCM) enables you to manage digital certificates and to use them in secure applications on the IBM i server. With Digital Certificate Manager, you can request and process digital certificates from Certificate Authorities (CAs) or other third-parties. You can also act as a local Certificate Authority to create and manage digital certificates for your users.

DCM also supports using Certificate Revocation Lists (CRLs) to provide a stronger certificate and application validation process. You can use DCM to define the location where a specific Certificate Authority CRL resides on an LDAP server so that IBM MQ can verify that a specific certificate has not been revoked.

DCM supports and can automatically detect certificates in a variety of formats. When DCM detects a PKCS #12 encoded certificate, or a PKCS #7 certificate that contains encrypted data, it automatically prompts the user to enter the password that was used to encrypt the certificate. DCM does not prompt for PKCS #7 certificates that do not contain encrypted data.

DCM provides a browser-based user interface that you can use to manage digital certificates for your applications and users. The user interface is divided into two main frames: a navigation frame and a task frame.

You use the navigation frame to select the tasks to manage certificates or the applications that use them. Some individual tasks are shown directly in the main navigation frame, but most tasks in the navigation frame are organized into categories. For example, Manage Certificates is a task category that contains various individual guided tasks, such as View certificate, Renew certificate, and Import certificate. If an item in the navigation frame is a category that contains more than one task, an arrow is displayed to the left of it. The arrow indicates that when you select the category link, an expanded list of tasks displays, enabling you to choose which task to perform.

For important information about DCM, see the following IBM Redbooks® publications:

- *IBM i Wired Network Security: OS/400 V5R1 DCM and Cryptographic Enhancements*, SG24-6168. Specifically, see the appendixes for essential information about setting up your IBM i system as a local CA.
- *AS/400 Internet Security: Developing a Digital Certificate Infrastructure*, SG24-5659. Specifically, see Chapter 5. *Digital Certificate Manager for AS/400* , which explains the AS/400 DCM.

Federal Information Processing Standards (FIPS)

This topic introduces the Federal Information Processing Standards (FIPS) Cryptomodule Validation Program of the US National Institute of Standards and Technology and the cryptographic functions which can be used on TLS channels.

Note: On AIX, Linux, and Windows, IBM MQ provides FIPS 140-2 compliance through the IBM Crypto for C (ICC) cryptographic module. The certificate for this module has been moved to the Historical status. Customers should view the IBM Crypto for C (ICC) certificate and be aware of any advice provided by NIST. A replacement FIPS 140-3 module is currently in progress and its status can be viewed by searching for it in the NIST CMVP modules in process list.

The IBM MQ Operator 3.2.0 and queue manager container image 9.4.0.0 onwards are based on UBI 9. FIPS 140-3 compliance is currently pending and its status can be viewed by searching for "Red Hat Enterprise Linux 9 - OpenSSL FIPS Provider" in the NIST CMVP modules in process list.

This information applies to the following platforms:

- ALW AIX, Linux, and Windows
- z/OS z/OS

ALW For more information about the FIPS 140-2 compliance of an IBM MQ TLS connection on AIX, Linux, and Windows, see “Federal Information Processing Standards (FIPS) for AIX, Linux, and Windows” on page 35.

z/OS For more information about the FIPS 140-2 compliance of an IBM MQ TLS connection on z/OS, see “Federal Information Processing Standards (FIPS) for z/OS” on page 37.

If cryptographic hardware is present, the cryptographic modules used by IBM MQ can be configured to be those provided by the hardware manufacturer. If this is done, the configuration is only FIPS-compliant if those cryptographic modules are FIPS-certified.

Over time, the Federal Information Processing Standards are updated to reflect new attacks against encryption algorithms and protocols. For example, some CipherSpecs may cease to be FIPS certified. When such changes occur, IBM MQ is also updated to implement the latest standard. As a result, you might see changes in behavior after applying maintenance.

Related concepts

“Specifying that only FIPS-certified CipherSpecs are used at run time on the MQI client” on page 264

Create your key repositories using FIPS-compliant software, then specify that the channel must use FIPS-certified CipherSpecs.

Related tasks

Enabling TLS in IBM MQ classes for Java

Using Transport Layer Security (TLS) with IBM MQ classes for JMS

Related reference

TLS properties of JMS objects

"runmqakm and runmqktool commands on AIX, Linux, and Windows" on page 528
On AIX, Linux, and Windows systems, use the **`runmqakm`** (GSKCapiCmd) or **`runmqktool`** (keytool) commands to manage keys and certificates.

"Federal Information Processing Standards" on page 23
The US government produces technical advice on IT systems and security, including data encryption. The National Institute for Standards and Technology (NIST) is an important body concerned with IT systems and security. NIST produces recommendations and standards, including the Federal Information Processing Standards (FIPS).

ALW *Federal Information Processing Standards (FIPS) for AIX, Linux, and Windows*
When cryptography is required on an SSL/TLS channel on AIX, Linux, and Windows systems, IBM MQ uses a cryptography package called IBM Crypto for C (ICC). On the AIX, Linux, and Windows platforms, the ICC software has passed the Federal Information Processing Standards (FIPS) Cryptomodule Validation Program of the US National Institute of Standards and Technology, at level 140-2.

Note: On AIX, Linux, and Windows, IBM MQ provides FIPS 140-2 compliance through the IBM Crypto for C (ICC) cryptographic module. The certificate for this module has been moved to the Historical status. Customers should view the IBM Crypto for C (ICC) certificate and be aware of any advice provided by NIST. A replacement FIPS 140-3 module is currently in progress and its status can be viewed by searching for it in the NIST CMVP modules in process list.

The IBM MQ Operator 3.2.0 and queue manager container image 9.4.0.0 onwards are based on UBI 9. FIPS 140-3 compliance is currently pending and its status can be viewed by searching for "Red Hat Enterprise Linux 9 - OpenSSL FIPS Provider" in the NIST CMVP modules in process list.

The FIPS 140-2 compliance of an IBM MQ TLS connection on AIX, Linux, and Windows systems is as follows:

- For all IBM MQ message channels (except CLNTCONN channel types), the connection is FIPS-compliant if the following conditions are met:
  - The installed IBM Global Security Kit (GSKit) ICC version has been certified FIPS 140-2 compliant on the installed operating system version and hardware architecture.
  - The queue manager's SSLFIPS attribute has been set to YES.
  - All key repositories have been created and manipulated using only FIPS-compliant software, such as **`runmqakm`** with the `-fips` option.
  - Access to all key repositories is provided using a stash file and not the queue manager's **KEYRPWD** attribute.
- For all IBM MQ MQI client applications , the connection uses GSKit and is FIPS-compliant if the following conditions are met:
  - The installed GSKit ICC version has been certified FIPS 140-2 compliant on the installed operating system version and hardware architecture.
  - You have specified that only FIPS-certified cryptography is to be used, as described in the related topic for the MQI client.
  - All key repositories have been created and manipulated using only FIPS-compliant software, such as **`runmqakm`** with the `-fips` option.
  - Access to all key repositories is provided using a stash file and not the key repository password mechanism.

- For IBM MQ classes for Java applications using client mode, the connection uses the JRE's TLS implementations and is FIPS-compliant if the following conditions are met:
  - The Java Runtime Environment used to run the application is FIPS-compliant on the installed operating system version and hardware architecture.
  - You have specified that only FIPS-certified cryptography is to be used, as described in the related topic for the Java client.
  - All key repositories have been created and manipulated using only FIPS-compliant software, such as **runmqakm** with the `-fips` option.
- For IBM MQ classes for JMS applications using client mode, the connection uses the JRE's TLS implementations and is FIPS-compliant if the following conditions are met:
  - The Java Runtime Environment used to run the application is FIPS-compliant on the installed operating system version and hardware architecture.
  - You have specified that only FIPS-certified cryptography is to be used, as described in the related topic for the JMS client.
  - All key repositories have been created and manipulated using only FIPS-compliant software, such as **runmqakm** with the `-fips` option.
- For unmanaged .NET client applications, the connection uses GSKit and is FIPS-compliant if the following conditions are met:
  - The installed GSKit ICC version has been certified FIPS 140-2 compliant on the installed operating system version and hardware architecture.
  - You have specified that only FIPS-certified cryptography is to be used, as described in the related topic for the .NET client.
  - All key repositories have been created and manipulated using only FIPS-compliant software, such as **runmqakm** with the `-fips` option.
  - Access to all key repositories is provided using a stash file and not the key repository password mechanism.
- For unmanaged XMS .NET client applications, the connection uses GSKit and is FIPS-compliant if the following conditions are met:
  - The installed GSKit ICC version has been certified FIPS 140-2 compliant on the installed operating system version and hardware architecture.
  - You have specified that only FIPS-certified cryptography is to be used, as described in the XMS .NET documentation.
  - All key repositories have been created and manipulated using only FIPS-compliant software, such as **runmqakm** with the `-fips` option.
  - Access to all key repositories is provided using a stash file and not the key repository password mechanism.

All supported platforms are FIPS 140-2 certified except as noted in the readme file included with each fix pack or refresh pack.

For TLS connections using GSKit, the component which is FIPS 140-2 certified is named *ICC*. It is the version of this component which determines GSKit FIPS compliance on any given platform. To determine the ICC version currently installed, run the **dspmqver -p 64 -v** command.

Here is an example extract of the **dspmqver -p 64 -v** output relating to ICC:

```
ICC
============
@(#)CompanyName:   IBM Corporation
@(#)LegalTrademarks: IBM
@(#)FileDescription: IBM Crypto for C-language
@(#)FileVersion:   8.0.0.0
@(#)LegalCopyright:  Licensed Materials - Property of IBM
@(#)         ICC
@(#)         (C) Copyright IBM Corp. 2002, 2024.
@(#)         All Rights Reserved. US Government Users
```

```
@(#)         Restricted Rights - Use, duplication or disclosure
@(#)         restricted by GSA ADP Schedule Contract with IBM Corp.
@(#)ProductName:   icc_8.0 (GoldCoast Build) 100415
@(#)ProductVersion:  8.0.0.0
@(#)ProductInfo:   10/04/15.03:32:19.10/04/15.18:41:51
@(#)CMVCInfo:
```

The NIST certification statement for GSKit ICC 8 (included in GSKit 8) can be found at the following address: Cryptographic Module Validation Program.

If cryptographic hardware is present, the cryptographic modules used by IBM MQ can be configured to be those provided by the hardware manufacturer. If this is done, the configuration is only FIPS-compliant if those cryptographic modules are FIPS-certified.

Triple DES restrictions enforced when operating in compliance with FIPS 140-2

When IBM MQ is configured to operate in compliance with FIPS 140-2, additional restrictions are enforced in relation to Triple DES (3DES) CipherSpecs. These restrictions enable compliance with the US NIST SP800-67 recommendation.

1. All parts of the Triple DES key must be unique.
2. No part of the Triple DES key can be a Weak, Semi-Weak, or Possibly-Weak key according to the definitions in NIST SP800-67.
3. No more than 32 GB of data can be transmitted over the connection before a secret key reset must occur. By default, IBM MQ does not reset the secret session key so this reset must be configured. Failure to enable secret key reset when using a Triple DES CipherSpec and FIPS 140-2 compliance results in the connection closing with error AMQ9288 after the maximum byte count is exceeded. For information about how to configure secret key reset, see "Resetting SSL and TLS secret keys" on page 455.

IBM MQ generates Triple DES session keys which already comply with rules 1 and 2. However, to satisfy the third restriction you must enable secret key reset when using Triple DES CipherSpecs in a FIPS 140-2 configuration. Alternatively, you can avoid using Triple DES.

Related concepts

"Specifying that only FIPS-certified CipherSpecs are used at run time on the MQI client" on page 264
Create your key repositories using FIPS-compliant software, then specify that the channel must use FIPS-certified CipherSpecs.

Related tasks

Enabling TLS in IBM MQ classes for Java

Using Transport Layer Security (TLS) with IBM MQ classes for JMS

Related reference

TLS properties of JMS objects

"runmqakm and runmqktool commands on AIX, Linux, and Windows" on page 528
On AIX, Linux, and Windows systems, use the **runmqakm** (GSKCapiCmd) or **runmqktool** (keytool) commands to manage keys and certificates.

"Federal Information Processing Standards" on page 23
The US government produces technical advice on IT systems and security, including data encryption. The National Institute for Standards and Technology (NIST) is an important body concerned with IT systems and security. NIST produces recommendations and standards, including the Federal Information Processing Standards (FIPS).

z/OS *Federal Information Processing Standards (FIPS) for z/OS*

When cryptography is required on an SSL/TLS channel on z/OS , IBM MQ uses a service called System SSL. The objective of System SSL is to provide the capability to execute securely in a mode designed to adhere to the Federal Information Processing Standards (FIPS) Cryptomodule Validation Program of the US National Institute of Standards and Technology, at level 140-2.

When implementing FIPS 140-2 compliant connections with IBM MQ TLS connections there are a number of points to consider:

- To enable IBM MQ message channels for FIPS-compliance, ensure the following conditions are met:
  - System SSL Security Level 3 FMID is installed and configured (see Planning to install IBM MQ).
  - System SSL modules are validated.
  - The queue manager's SSLFIPS attribute has been set to **YES**.

When executing in FIPS mode, System SSL exploits CP Assist for Cryptographic Function (CPACF) when available. Cryptographic functions performed by ICSF-supported hardware when running in non-FIPS mode continue to be exploited when executing in FIPS mode, with the exception of RSA signature generation which must be performed in software.

Table 2. Differences between FIPS mode and non-FIPS mode algorithm support.

	Non-FIPS		**FIPS**	
Algorithm	**Key sizes**	**Hardware**	**Key sizes**	**Hardware**
RC2	40 and 128			
RC4	40 and 128			
DES	56	x		
TDES	168	x	168	x
AES	128 and 256	x	128 and 256	x
MD5	48			
SHA-1	160	x	160	x
SHA-2	224, 256, 384 and 512	x	224, 256, 384 and 512	x
RSA	512-4096	x	1024-4096	x
DSA	512-1024		1024	
DH	512-2048		2048	

In FIPS mode, System SSL can only use certificates that use the algorithms and key sizes shown in Table 1. During X.509 certificate validation if an algorithm that is incompatible with FIPS mode is encountered, then the certificate cannot be used and is treated as not valid.

For IBM MQ classes applications using client mode within WebSphere® Application Server , refer to Federal Information Processing Standard support.

For information on System SSL module configuration, see System SSL Module Verification Setup.

Related reference

"Federal Information Processing Standards" on page 23
The US government produces technical advice on IT systems and security, including data encryption. The National Institute for Standards and Technology (NIST) is an important body concerned with IT systems and security. NIST produces recommendations and standards, including the Federal Information Processing Standards (FIPS).

Multi *Verifying the TLS configuration of your queue manager with `mqcertck`*

The **`MQCERTCK`** command is a tool to look for common mistakes in the TLS configuration of your queue manager, and provides some suggestions for resolving problems.

Introduction

The **`mqcertck`** command checks the:

- Existence and permissions of the key repository of the queue manager, referenced in the queue manager **SSLKEYR** attribute.
- Existence and validity of the certificate for the queue manager certificate, referenced in the queue manager **CERTLABL** attribute.
- Existence and validity of any certificates referenced in the **CERTLABL** attributes of the TLS enabled channel.
- Key repository and certificates of the client applications, including checking the certificates are authorized with the queue manager.

Note: The **mqcertck** command is not available on z/OS or IBM i.

Usage

To use the **mqcertck** command, run the command `mqcertck`, together with its required parameters, and any optional parameters you require, from a command line.

See mqcertck for a description of the command and the parameters the command takes.

Example

You have just finished setting up your queue manager QM1 to allow TLS connections from clients connecting to the SVRCONN channel of your queue manager.

You are using the multiple certificates feature and so both your queue manager and channel have a certificate label specified in their **CERTLABL** attributes. While creating the channel you made a mistake in the **CERTLABL** attribute of the channel, so when a client attempts to connect, the queue manager returns a 2393 return code of MQRC_SSL_INITIALIZATION_ERROR.

Before activating the queue manager, you use the **mqcertck** command to verify the TLS configuration of the queue manager.

You run the command `mqcertck QM1` and receive the following output:

```
5724-H72 (C) Copyright IBM Corp. 1994, 2024.
+-----------------------------------------------------------
| IBM MQ TLS Configuration Test tool
+-----------------------------------------------------------
| Problem identified:
|  No certificate could be found for the channel
|  MQCERTCK.CHANNEL
|  This tool looked in the Queue Manager's key repository
|  located at: 'C:\MQ Data\qmgrs\QM1\ssl\key.kdb'
|  for a certificate with label 'chacert',
|  which is the certificate specified in the channel's
|  CERTLABL attribute, but was unable to find one.
|
| Possible resolution:
|  A valid certificate with the label chacert
|  needs to be added to the key repository.
|
|  Alternatively, alter the channel definition to remove
|  the CERTLABL value. This can be done by executing the
|  following command in runmqsc:
|      ALTER CHANNEL(<Name>) CHLTYPE(<TYPE>) CERTLABL(' ')
+-----------------------------------------------------------
| mqcertck has ended. See above for any problems found.
| If there are problems then resolve these and run this
| tool again.
|
+-----------------------------------------------------------
```

This output prompts you to check your channel definition for the server connection channel MQCERTCK.CHANNEL. Here, you see the error you made, and can correct the error before running the `mqcertck` command again to verify that you have resolved the problem.

Verifying client connections

The **mqcertck** command has the ability to verify client key repositories, as well as the TLS configuration of the queue manager. To do this, **mqcertck** needs to be able to access the key repository of the client from the machine running the queue manager.

When running the **mqcertck** command, if you supply the **-clientkeyr** parameter with the location of the client key repository (excluding the extension) **mqcertck** checks this key repository against the queue manager.

If you know which channel the client will be using to connect to the queue manager, you can specify this with the **-clientchannel** flag.

If the client is using mutual authentication to connect to the queue manager you can use the **-clientusername** or **-clientlabel** parameter, to tell the **mqcertck** command which certificate to use in the client key repository.

If you are using the default certificate, and not supplying a certificate label to the client application, you can use **-clientusername** and the **username** parameters which run this application.

During the operation of the **mqcertck** command, the command generates the certificate label `ibmwebspheremq`XXXX where XXXX is the value passed in the **-clientusername** parameter.

In order to fully verify the client key repository, the **mqcertck** command creates a dummy connection using IBM Global Security Kit (GSKit). To do this, the command needs to have a port available that it can bind to during its client tests. The default port used is 5857, however, if this is already in use you can specify a different port to be used during the client tests.

Note: Although the **mqcertck** command binds to a port, no external communications are used by **mqcertck**, and all tests are performed locally.

SSL/TLS on the IBM MQ MQI client

IBM MQ supports TLS on clients. You can tailor the use of TLS in various ways.

IBM MQ provides TLS support for IBM MQ MQI clients on AIX, Linux, and Windows systems. If you are using IBM MQ classes for Java, see Using IBM MQ classes for Java and if you are using IBM MQ classes for JMS, see Using IBM MQ classes for JMS. The rest of this section does not apply to the Java or JMS environments.

You can specify the key repository for an IBM MQ MQI client either with the MQSSLKEYR value in your IBM MQ client configuration file, or when your application makes an MQCONNX call. You have three options for specifying that a channel uses TLS:

- Using a channel definition table
- Using the SSL configuration options structure, MQSCO, on an MQCONNX call
- Using the Active Directory (on Windows systems)

You cannot use the MQSERVER environment variable to specify that a channel uses TLS.

You can continue to run your existing IBM MQ MQI client applications without TLS, as long as TLS is not specified at the other end of the channel.

If changes are made on a client machine to the contents of the TLS Key Repository, the location of the TLS Key Repository, the Authentication Information, or the Cryptographic hardware parameters, you need to end all the TLS connections in order to reflect these changes in the client-connection channels that the application is using to connect to the queue manager. Once all the connections have ended, restart the TLS channels. All the new TLS settings are used. These settings are analogous to those refreshed by the REFRESH SECURITY TYPE(SSL) command on queue manager systems.

When your IBM MQ MQI client runs on a AIX, Linux, and Windows system with cryptographic hardware, you configure that hardware with the MQSSLCRYP environment variable. This variable is equivalent to the SSLCRYP parameter on the ALTER QMGR MQSC command. Refer to ALTER QMGR for a description of the SSLCRYP parameter on the ALTER QMGR MQSC command. If you use the GSK_PCS11 version of the SSLCRYP parameter, the PKCS #11 token label must be specified entirely in lower-case.

TLS secret key reset and FIPS are supported on IBM MQ MQI clients. For more information, see "Resetting SSL and TLS secret keys" on page 455 and "Federal Information Processing Standards (FIPS) for AIX, Linux, and Windows" on page 35.

See "Setting up IBM MQ MQI client security" on page 263 for more information about the TLS support for IBM MQ MQI clients.

Related tasks

IBM MQ MQI client configuration file, `mqclient.ini`

Specifying that an MQI channel uses SSL/TLS

For an MQI channel to use TLS, the value of the *SSLCipherSpec* attribute of the client-connection channel must be the name of a CipherSpec that is supported by IBM MQ on the client platform.

You can define a client-connection channel with a value for this attribute in the following ways. They are listed in order of decreasing precedence.

1. When a PreConnect exit provides a channel definition structure to use.

   A PreConnect exit can provide the name of a CipherSpec in the *SSLCipherSpec* field of a channel definition structure, MQCD. This structure is returned in the **`ppMQCDArrayPtr`** field of the MQNXP exit parameter structure used by the PreConnect exit.
2. When an IBM MQ MQI client application issues an MQCONNX call.

   The application can specify the name of a CipherSpec in the *SSLCipherSpec* field of a channel definition structure, MQCD. This structure is referenced by the connect options structure, MQCNO, which is a parameter on the MQCONNX call.
3. Using a client channel definition table (CCDT).

   One or more entries in a client channel definition table can specify the name of a CipherSpec. For example, if you create an entry by using the DEFINE CHANNEL MQSC command, you can use the SSLCIPH parameter on the command to specify the name of a CipherSpec.
4. Using Active Directory on Windows.

   On Windows systems, you can use the **`setmqscp`** control command to publish the client-connection channel definitions in Active Directory. One or more of these definitions can specify the name of a CipherSpec.

For example, if a client application provides a client-connection channel definition in an MQCD structure on an MQCONNX call, this definition is used in preference to any entries in a client channel definition table that can be accessed by the IBM MQ client.

You cannot use the MQSERVER environment variable to provide the channel definition at the client end of an MQI channel that uses TLS.

To check whether a client certificate has flowed, display the channel status at the server end of a channel for the presence of a peer name parameter value.

Related concepts

"Specifying a CipherSpec for an IBM MQ MQI client" on page 433

You have three options for specifying a CipherSpec for an IBM MQ MQI client.

CipherSpecs and CipherSuites in IBM MQ

IBM MQ supports TLS1.3 and TLS 1.2 CipherSpecs, and RSA and Diffie-Hellman algorithms. However, you can enable deprecated CipherSpecs, if you need to do so.

See "Enabling CipherSpecs" on page 411 for information on:

- CipherSpecs supported by IBM MQ.
- How you enable deprecated SSL 3.0 and TLS 1.0 CipherSpecs.

IBM MQ supports the RSA and Diffie-Hellman key exchange and authentication algorithms. The size of the key used during the TLS handshake can depend on the digital certificate you use, but some

CipherSpecs include a specification of the handshake key size. Larger handshake key sizes provide stronger authentication. With smaller key sizes, the handshake is faster.

Related concepts

"CipherSpecs and CipherSuites" on page 22
Cryptographic security protocols must agree on the algorithms used by a secure connection. CipherSpecs and CipherSuites define specific combinations of algorithms.

NSA Suite B Cryptography in IBM MQ

This topic provides information about how to configure IBM MQ for AIX, Linux, and Windows to conform to the Suite B compliant TLS 1.2 profile.

Over time, the NSA Cryptography Suite B Standard is updated to reflect new attacks against encryption algorithms and protocols. For example, some CipherSpecs might cease to be Suite B certified. When such changes occur, IBM MQ is also updated to implement the latest standard. As a result, you might see changes in behavior after applying maintenance. The IBM MQ readme file lists the version of Suite B enforced by each product maintenance level. If you configure IBM MQ to enforce Suite B compliance, always consult the readme file when planning to apply maintenance. See IBM MQ, WebSphere MQ, and MQSeries® product readmes.

On AIX, Linux, and Windows systems, IBM MQ can be configured to conform to the Suite B compliant TLS 1.2 profile at the security levels shown in Table 1.

Table 3. Suite B security levels with allowed CipherSpecs and digital signature algorithms

Security level	Allowed CipherSpecs	Allowed digital signature algorithms
128-bit	ECDHE_ECDSA_AES_128_GCM_SHA256 ECDHE_ECDSA_AES_256_GCM_SHA384	ECDSA with SHA-256 ECDSA with SHA-384
192-bit	ECDHE_ECDSA_AES_256_GCM_SHA384	ECDSA with SHA-384
Both [1]	ECDHE_ECDSA_AES_128_GCM_SHA256 ECDHE_ECDSA_AES_256_GCM_SHA384	ECDSA with SHA-256 ECDSA with SHA-384

1. It is possible to configure both the 128-bit and 192-bit security levels concurrently. Since the Suite B configuration determines the minimum acceptable cryptographic algorithms, configuring both security levels is equivalent to configuring only the 128-bit security level. The cryptographic algorithms of the 192-bit security level are stronger than the minimum required for the 128-bit security level, so they are permitted for the 128-bit security level even if the 192-bit security level is not enabled.

Note: The naming conventions used for the Security level do not necessarily represent the elliptic curve size or the key size of the AES encryption algorithm.

CipherSpec conformation to Suite B

Although the default behavior of IBM MQ is not to comply with the Suite B standard, IBM MQ can be configured to conform to either, or both security levels on AIX, Linux, and Windows systems. Following the successful configuration of IBM MQ to use Suite B, any attempt to start an outbound channel using a CipherSpec not conforming to Suite B results in the error `AMQ9282`. This activity also results in the MQI client returning the reason code `MQRC_CIPHER_SPEC_NOT_SUITE_B`. Similarly, attempting to start an inbound channel using a CipherSpec not conforming to the Suite B configuration results in the error `AMQ9616`.

For more information about IBM MQ CipherSpecs, see "Enabling CipherSpecs" on page 411

Suite B and digital certificates

Suite B restricts the digital signature algorithms which can be used to sign digital certificates. Suite B also restricts the type of public key which certificates can contain. Therefore IBM MQ must be configured to use certificates whose digital signature algorithm and public key type are allowed by the configured Suite B security level of the remote partner. Digital certificates which do not comply with the security level requirements are rejected and the connection fails with error `AMQ9633` or `AMQ9285`.

For the 128-bit Suite B security level, the public key of the certificate subject is required to use either the NIST P-256 elliptic curve or the NIST P-384 elliptic curve and to be signed with either the NIST P-256 elliptic curve or the NIST P-384 elliptic curve. At the 192-bit Suite B security level, the public key of the certificate subject is required to use the NIST P-384 elliptic curve and to be signed with the NIST P-384 elliptic curve.

To obtain a certificate suitable for Suite B compliant operation, use the **`runmqakm`** command and specify the **`-sig_alg`** parameter to request a suitable digital signature algorithm. The `EC_ecdsa_with_SHA256` and `EC_ecdsa_with_SHA384` **`-sig_alg`** parameter values correspond to elliptic curve keys signed by the allowed Suite B digital signature algorithms.

For more information about the **`runmqakm`** command, see "Managing keys and certificates on AIX, Linux, and Windows" on page 527.

Creating and requesting digital certificates

To create a self-signed digital certificate for Suite B testing, see "Creating a self-signed personal certificate on AIX, Linux, and Windows" on page 529

To request a CA-signed digital certificate for Suite B production use, see "Requesting a personal certificate on AIX, Linux, and Windows" on page 531.

Note: The certificate authority being used must generate digital certificates which satisfy the requirements described in IETF RFC 6460.

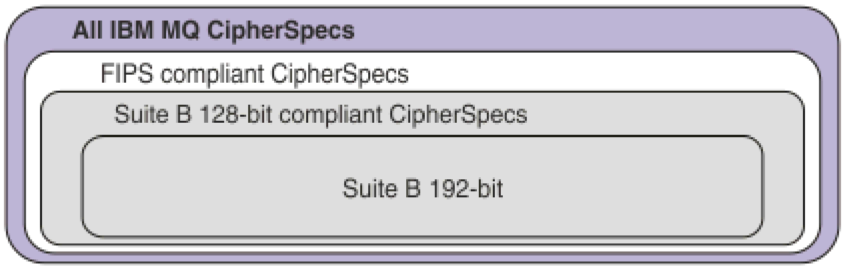
FIPS 140-2 and Suite B

Note: On AIX, Linux, and Windows, IBM MQ provides FIPS 140-2 compliance through the IBM Crypto for C (ICC) cryptographic module. The certificate for this module has been moved to the Historical status. Customers should view the IBM Crypto for C (ICC) certificate and be aware of any advice provided by NIST. A replacement FIPS 140-3 module is currently in progress and its status can be viewed by searching for it in the NIST CMVP modules in process list.

The IBM MQ Operator 3.2.0 and queue manager container image 9.4.0.0 onwards are based on UBI 9. FIPS 140-3 compliance is currently pending and its status can be viewed by searching for "Red Hat Enterprise Linux 9 - OpenSSL FIPS Provider" in the NIST CMVP modules in process list.

The Suite B standard is conceptually similar to FIPS 140-2, as it restricts the set of enabled cryptographic algorithms in order to provide an assured level of security. The Suite B CipherSpecs currently supported can be used when IBM MQ is configured for FIPS 140-2 compliant operation. It is therefore possible to configure IBM MQ for both FIPS and Suite B compliance simultaneously, in which case both sets of restrictions apply.

The following diagram illustrates the relationship between these subsets:


All IBM MQ CipherSpecs
FIPS compliant CipherSpecs
Suite B 128-bit compliant CipherSpecs
Suite B 192-bit

Configuring IBM MQ for Suite B compliant operation

For information about how to configure IBM MQ on AIX, Linux, and Windows for Suite B compliant operation, see "Configuring IBM MQ for Suite B" on page 44.

IBM MQ does not support Suite B compliant operation on the following platforms and clients:

- IBM i platform
- z/OS platform
- Java client
- JMS client

Related concepts

"Specifying that only FIPS-certified CipherSpecs are used at run time on the MQI client" on page 264
Create your key repositories using FIPS-compliant software, then specify that the channel must use FIPS-certified CipherSpecs.

ALW *Configuring IBM MQ for Suite B*

IBM MQ can be configured to operate in compliance with the NSA Suite B standard on AIX, Linux, and Windows platforms.

Suite B restricts the set of enabled cryptographic algorithms in order to provide an assured level of security. IBM MQ can be configured to operate in compliance with Suite B to provide an enhanced level of security. For further information on Suite B, see "National Security Agency (NSA) Suite B Cryptography" on page 24. For more information about Suite B configuration and its effect on TLS channels, see "NSA Suite B Cryptography in IBM MQ" on page 42.

Queue manager

For a queue manager, use the command **ALTER QMGR** with the parameter **SUITEB** to set the values appropriate for your required level of security. For more information see ALTER QMGR.

You can also use the PCF **MQCMD_CHANGE_Q_MGR** command with the **MQIA_SUITE_B_STRENGTH** parameter to configure the queue manager for Suite B compliant operation.

Note: If you alter a queue manager's Suite B settings, you must restart the MQXR service for those settings to take effect.

MQI client

By default, MQI clients do not enforce Suite B compliance. You can enable the MQI client for Suite B compliance by executing one of the following options:

1. By setting the EncryptionPolicySuiteB field in the MQSCO structure on an MQCONNX call to one or more of the following values:

   - `MQ_SUITE_B_NONE`
   - `MQ_SUITE_B_128_BIT`
   - `MQ_SUITE_B_192_BIT`

   Using `MQ_SUITE_B_NONE` with any other value is invalid.

   For more information about the MQSCO structure, see MQSCO - SSL configuration options.
2. By setting the **MQSUITEB** environment variable to one or more of the following values:

   - `NONE`
   - `128_BIT`
   - `192_BIT`

   You can specify multiple values using a comma separated list. Using the value `NONE` with any other value is invalid.

3. By setting the **EncryptionPolicySuiteB** attribute in the SSL stanza of the client configuration file to one or more of the following values:
   - NONE
   - 128_BIT
   - 192_BIT

   You can specify multiple values using a comma separated list. Using NONE with any other value is invalid.

Note: The MQI client settings are listed in order of priority. The MSCO structure on the MQCONNX call overrides the setting on the **MQSUITEB** environment variable, which overrides the attribute in the SSL stanza.

.NET

For .NET unmanaged clients, the property **MQC.ENCRYPTION_POLICY_SUITE_B** indicates the type of Suite B security required.

For information about the using Suite B in IBM MQ classes for .NET, see MQEnvironment .NET class.

AMQP

The Suite B attribute settings for a queue manager apply to AMQP channels on that queue manager. If you modify the queue manager Suite B settings, you must restart the AMQP service for the changes to take effect.

Certificate validation policies in IBM MQ

The certificate validation policy determines how strictly the certificate chain validation conforms to industry security standards.

The certificate validation policy depends upon the platform and environment as follows:

- For Java and JMS applications on all platforms, the certificate validation policy depends on the JSSE component of the Java runtime environment. For more information about the certificate validation policy, see the documentation for your JRE.
- ALW For AIX, Linux, and Windows systems, the certificate validation policy is supplied by IBM Global Security Kit (GSKit) and can be configured. V 9.4.0 V 9.4.0 Three different certificate validation policies are supported:
  - A legacy certificate validation policy, used for maximum backwards compatibility and interoperability with old digital certificates that do not comply with the current IETF certificate validation standards. This policy is known as the Basic policy.
  - A strict, standards-compliant certificate validation policy which enforces the RFC 5280 standard. This policy is known as the Standard policy.
  - V 9.4.0 V 9.4.0 A certificate validation policy which does not authenticate the TLS server certificate, available only for client applications.
- IBM i For IBM i systems, the certificate validation policy depends on the secure sockets library provided by the operating system. For more information about the certificate validation policy, see the documentation for the operating system.
- z/OS For z/OS systems, the certificate validation policy depends on the System SSL component provided by the operating system. For more information about the certificate validation policy, see the documentation for the operating system.

For information about how to configure the certificate validation policy, see "Configuring certificate validation policies in IBM MQ" on page 46. For more information about the differences between the

Basic and Standard certificate validation policies, see Certificate validation and trust policy design on AIX, Linux, and Windows.

Configuring certificate validation policies in IBM MQ

There are several different ways in which you can specify which TLS certificate validation policy is used to validate digital certificates received from remote partner systems.

About this task

The certificate validation policy determines how strictly the certificate chain validation conforms to industry security standards. The certificate validation policy depends upon the platform and environment. For more information about certificate validation policies, see "Certificate validation policies in IBM MQ" on page 45.

Procedure

- To set the certificate validation policy on the queue manager, use the queue manager attribute **CERTVPOL**.

  For more information about setting this attribute, see ALTER QMGR (alter queue manager settings).
- To set the certificate validation policy on the client, use of the following methods.

  If more than one method is used to set the policy, the client uses the settings in the following priority order:

  1. Use the `CertificateValPolicy` field in the client MQSCO structure. Set the field to one of the following values:

     MQ_CERT_VAL_POLICY_ANY
     Apply each of the certificate validation policies supported by the secure sockets library. Accept the certificate chain if any of the policies considers the certificate chain valid.

     MQ_CERT_VAL_POLICY_RFC5280
     Apply only the RFC5280 compliant certificate validation policy. This setting provides stricter validation than the ANY setting, but rejects some older digital certificates.

     V 9.4.0 V 9.4.0 **MQ_CERT_VAL_POLICY_NONE**
     Apply no certificate validation policy. This setting is for client applications only and accepts the TLS server certificate without validating the trust chain.

     For more information about using this field, see MQSCO - SSL configuration options.
  2. Use the client environment variable **MQCERTVPOL**. To set this environment variable, use one of the following commands:

     – Linux AIX For AIX and Linux systems:

     ```
     export MQCERTVPOL= value
     ```

     – Windows For Windows systems:

     ```
     SET MQCERTVPOL= value
     ```

     – IBM i For IBM i systems:

     ```
     ADDENVVAR ENVVAR(MQCERTVPOL) VALUE(value)
     ```
  3. Use the **CertificateValPolicy** attribute of the SSL stanza in the client configuration file. Set this attribute to one of the following values:

ANY

Use any certificate validation policy supported by the underlying secure sockets library. This setting is the default setting.

RFC5280

Use only certificate validation which complies with the RFC 5280 standard.

V 9.4.0 V 9.4.0 **NONE**

Apply no certificate validation policy. This setting accepts the TLS server certificate without validating the trust chain.

For more information about using this attribute, see SSL stanza of the client configuration file.

Digital certificates and CipherSpec compatibility in IBM MQ

This topic provides information on how to choose appropriate CipherSpecs and digital certificates for your security policy, by outlining the relationship between CipherSpecs and digital certificates in IBM MQ.

Only a subset of the supported CipherSpecs can be used with all of the supported types of digital certificate. It is therefore necessary to choose an appropriate CipherSpec for your digital certificate. Similarly, if your organization's security policy requires that you use a particular CipherSpec then you must obtain an appropriate digital certificate for that CipherSpec.

The MD5 digital signature algorithm and TLS 1.2

Digital certificates signed using the MD5 algorithm are rejected when the TLS 1.2 protocol is used. This is because the MD5 algorithm is now considered weak by many cryptographic analysts and its use is generally discouraged. To use newer CipherSpecs based on the TLS 1.2 protocol, ensure that the digital certificates do not use the MD5 algorithm in their digital signatures. Older CipherSpecs which use the TLS 1.0 protocols are not subject to this restriction and can continue to use certificates with MD5 digital signatures.

To view the digital signature algorithm for a particular certificate, you can use the **runmqakm** command:

```
runmqakm -cert -details -db key.kdb -pw password -label cert_label
```

where *cert_label* is the certificate label of the digital signature algorithm to to display. See Digital certificate labels for details.

Running the **runmqakm** command produces output displaying the use of the signature algorithm specified:

```
Label : ibmmqexample
Key Size : 1024
Version : X509 V3
Serial : 4e4e93f1
Issuer : CN=Old Certificate Authority,OU=Test,O=Example,C=US
Subject : CN=Example Queue Manager,OU=Test,O=Example,C=US
Not Before : August 19, 2011 5:48:49 PM GMT+01:00
Not After : August 18, 2012 5:48:49 PM GMT+01:00
Public Key
    30 81 9F 30 0D 06 09 2A 86 48 86 F7 0D 01 01 01
    05 00 03 81 8D 00 30 81 89 02 81 81 00 98 5A 7A
    F0 18 21 EE E4 8A 6E DE C8 01 4B 3A 1E 41 90 3D
    CE 01 3F E6 32 30 6C 23 59 F0 FE 78 6D C2 80 EF
    BC 83 54 7A EB 60 80 62 6B F1 52 FE 51 9D C1 61
    80 A5 1C D4 F0 76 C7 15 6D 1F 0D 4D 31 3E DC C6
    A9 20 84 6E 14 A1 46 7D 4C F5 79 4D 37 54 0A 3B
    A9 74 ED E7 8B 0F 80 31 63 1A 0B 20 A5 99 EE 0A
    30 A6 B6 8F 03 97 F6 99 DB 6A 58 89 7F 27 34 DE
    55 08 29 D8 A9 6B 46 E6 02 17 C3 13 D3 02 03 01
    00 01
Public Key Type : RSA (1.2.840.113549.1.1.1)
Fingerprint : SHA1 :
    09 4E 4F F2 1B CB C1 F4 4F 15 C9 2A F7 32 0A 82
    DA 45 92 9F
Fingerprint : MD5 :
    44 54 81 7C 58 68 08 3A 5D 75 96 40 D5 8C 7A CB
Fingerprint : SHA256 :
    3B 47 C6 E7 7B B0 FF 85 34 E7 48 BE 11 F2 D4 35
    B7 9A 79 53 2B 07 F5 E7 65 E8 F7 84 E0 2E 82 55
Signature Algorithm : MD5WithRSASignature (1.2.840.113549.1.1.4)
Value
    3B B9 56 E6 F2 77 94 69 5B 3F 17 EA 7B 19 D0 A2
    D7 10 38 F1 88 A4 44 1B 92 35 6F 3B ED 99 9B 3A
    A5 A4 FC 72 25 5A A9 E3 B1 96 88 FC 1E 9F 9B F1
    C5 E8 8E CF C4 8F 48 7B 0E A6 BB 13 AE 2B BD D8
    63 2C 03 38 EF DC 01 E1 1F 7A 6F FB 2F 65 74 D0
    FD 99 94 BA B2 3A D5 B4 89 6C C1 2B 43 6D E2 39
    66 6A 65 CB C3 C4 E2 CC F5 49 39 A3 8B 93 5A DD
    B0 21 0B A8 B2 59 5B 24 59 50 44 89 DC 78 19 51
Trust Status : Enabled
```

The `Signature Algorithm` line shows that the `MD5WithRSASignature` algorithm is used. This algorithm is based upon MD5 and so this digital certificate cannot be used with the TLS 1.2 CipherSpecs.

Interoperability of Elliptic Curve and RSA CipherSpecs

Not all CipherSpecs can be used with all digital certificates. CipherSpecs are denoted by the CipherSpec name prefix. Each type of CipherSpec imposes different restrictions upon the type of digital certificate which can be used. These restrictions apply to all IBM MQ TLS connections, but are particularly relevant to users of Elliptic Curve cryptography.

The following table summarizes the relationships between CipherSpecs and digital certificates:

Table 4. Relationships between CipherSpecs and digital certificates

Type	CipherSpec Name Prefix	Description	Required public key type	Digital signature encryption algorithm	Secret key establishment method
1	ECDHE_ECDSA_	CipherSpecs which use Elliptic Curve public keys, Elliptic Curve secret keys, and Elliptic Curve digital signature algorithms.	Elliptic Curve	ECDSA	ECDHE
2	ECDHE_RSA_	CipherSpecs which use RSA public keys, Elliptic Curve secret keys, and RSA digital signature algorithms.	RSA	RSA	ECDHE
3	(All TLS 1.3 CipherSpecs)	CipherSpecs which use Elliptic Curve or RSA public keys, Elliptic Curve secret keys, and Elliptic Curve or RSA digital signature algorithms.	Elliptic Curve or RSA	ECDSA or RSA	ECDHE or RSA
4	(All others)	CipherSpecs which use RSA public keys and RSA digital signature algorithms.	RSA	RSA	RSA

Note: Type 1 and 2 CipherSpecs are not supported by IBM MQ queue managers and MQI clients on the IBM i platform.

The required public key type column shows the type of public key which the personal certificate must have when using each type of CipherSpec. The personal certificate is the end-entity certificate which identifies the queue manager or client to its remote partner.

You must ensure that the certificate that is named in the certificate label is appropriate for the channel CipherSpec. That is, if you configure a channel with a CipherSpec that requires an Elliptic Curve (EC) certificate, you cannot name an RSA certificate in the certificate label. If you configure a channel with a CipherSpec that requires an RSA certificate, you cannot name an EC certificate in the certificate label.

Assuming that you have correctly configured IBM MQ, you can have:

- A single queue manager with a mixture of RSA and EC certificates.
- Different channels on the same queue manager using either an RSA or EC certificate.

The digital signature encryption algorithm refers to the encryption algorithm used to validate the peer. The encryption algorithm is used along with a hash algorithm such as MD5, SHA-1 or SHA-256 to compute the digital signature. There are various digital signature algorithms which can be used, for example, RSA with MD5 or ECDSA with SHA-256. In the table, ECDSA refers to the set of digital signature algorithms which use ECDSA; RSA refers to the set of digital signature algorithms which use RSA. Any supported digital signature algorithm in the set may be used, provided it is based upon the stated encryption algorithm.

Type 1 CipherSpecs require that the personal certificate must have an Elliptic Curve public key. When these CipherSpecs are used, Elliptic Curve Diffie Hellman Ephemeral key agreement is used to establish the secret key for the connection.

Type 2 CipherSpecs require that the personal certificate has an RSA public key. When these CipherSpecs are used, Elliptic Curve Diffie Hellman Ephemeral key agreement is used to establish the secret key for the connection.

Type 3 CipherSpecs require that the personal certificate must have an RSA public key. When these CipherSpecs are used, RSA key exchange is used to establish the secret key for the connection.

This list of restrictions is not exhaustive: depending on the configuration, there might be additional restrictions which can further affect the ability to interoperate. For example, if IBM MQ is configured to comply with the FIPS 140-2 or NSA Suite B standards then this will also limit the range of allowable configurations. Refer to the following section for more information.

If you need to use different types of CipherSpec on the same queue manager or client application, configure an appropriate certificate label and CipherSpec combination on the client definition.

The three types of CipherSpec do not interoperate directly: this is a limitation of the current TLS standards. For example, suppose you have chosen to use the ECDHE_ECDSA_AES_128_CBC_SHA256 CipherSpec for a receiver channel named TO.QM1 on a queue manager named QM1, then the receiver should have a personal certificate with an Elliptic Curve key and an ECDSA-based digital signature. If the receiver channel does not meet these requirements, the channel fails to start.

Other channels connecting to queue manager QM1 can use other CipherSpecs, provided that each channel uses a certificate of the correct type for the CipherSpec of that channel. For example, suppose that QM1 uses a sender channel named TO.QM2 to send messages to another queue manager named QM2. The channel TO.QM2 could use the Type 3 CipherSpec TLS_RSA_WITH_AES_256_CBC_SHA256 provided both ends of the channel use certificates containing RSA public keys. The certificate label channel attribute can be used to configure a different certificate for each channel.

When planning your IBM MQ networks, consider carefully which channels require TLS, and ensure that the type of certificates used for each channel are appropriate for use with the CipherSpec on that channel.

To view the digital signature algorithm and public key type for a digital certificate you can use the **runmqakm** command:

```
runmqakm -cert -details -db key.kdb -pw password -label cert_label
```

where *cert_label* is the label of the certificate whose digital signature algorithm you need to display. See Digital certificate labels for details.

Running the **runmqakm** command will produce output displaying the Public Key Type:

```
Label : ibmmqexample
Key Size : 384
Version : X509 V3
Serial : 9ad5eeef5d756f41
Issuer : CN=Example Certificate Authority,OU=Test,O=Example,C=US
Subject : CN=Example Queue Manager,OU=Test,O=Example,C=US
```

```
Not Before : 21 August 2011 13:10:24 GMT+01:00
Not After : 21 August 2012 13:10:24 GMT+01:00
Public Key
    30 76 30 10 06 07 2A 86 48 CE 3D 02 01 06 05 2B
    81 04 00 22 03 62 00 04 3E 6F A9 06 B6 C3 A0 11
    F8 D6 22 78 FE EF 0A FE 34 52 C0 8E AB 5E 81 73
    D0 97 3B AB D6 80 08 E7 31 E9 18 3F 6B DE 06 A7
    15 D6 9D 5B 6F 56 3B 7F 72 BB 6F 1E C9 45 1C 46
    60 BE F2 DC 1B AD AC EC 64 4C 0E 06 65 6E ED 93
    B8 F5 95 E0 F9 2A 05 D6 21 02 BD FB 06 63 A1 CC
    66 C6 8A 0A 5C 3F F7 D3
Public Key Type : EC_ecPublicKey (1.2.840.10045.2.1)
Fingerprint : SHA1 :
    3C 34 58 04 5B 63 5F 5C C9 7A E7 67 08 2B 84 43
    3D 43 7A 79
Fingerprint : MD5 :
    49 13 13 E1 B2 AC 18 9A 31 41 DC 8C B4 D6 06 68
Fingerprint : SHA256 :
    6F 76 78 68 F3 70 F1 53 CE 39 31 D9 05 C5 C5 9F
    F2 B8 EE 21 49 16 1D 90 64 6D AC EB 0C A7 74 17
Signature Algorithm : EC_ecdsa_with_SHA384 (1.2.840.10045.4.3.3)
Value
    30 65 02 30 0A B0 2F 72 39 9E 24 5A 22 FE AC 95
    0D 0C 6D 6C 2F B3 E7 81 F6 C1 36 1B 9A B0 6F 07
    59 2A A1 4C 02 13 7E DD 06 D6 FE 4B E4 03 BC B1
    AC 49 54 1E 02 31 00 90 0E 46 2B 04 37 EE 2C 5F
    1B 9C 69 E5 99 60 84 84 10 71 1A DA 63 88 33 E2
    22 CC E6 1A 4E F4 61 CC 51 F9 EE A0 8E F4 DC B5
    0B B9 72 58 C3 C7 A4
Trust Status : Enabled
```

The Public Key Type line in this case shows that the certificate has an Elliptic Curve public key. The Signature Algorithm line in this case shows that the EC_ecdsa_with_SHA384 algorithm is in use: this is based upon the ECDSA algorithm. This certificate is therefore only suitable for use with Type 1 CipherSpecs.

TLS 1.3 CipherSpecs

TLS 1.3 CipherSpecs support both ECDSA and RSA certificates.

Elliptic Curve CipherSpecs and NSA Suite B

When IBM MQ is configured to conform to the Suite B compliant TLS 1.2 profile, the permitted CipherSpecs and digital signature algorithms are restricted as described in "NSA Suite B Cryptography in IBM MQ" on page 42. Additionally, the range of acceptable Elliptic Curve keys is reduced according to the configured security levels.

At the 128-bit Suite B security level, the certificate subject's public key is required to use either the NIST P-256 or NIST P-384 elliptic curve and to be signed with either the NIST P-256 elliptic curve or the NIST P-384 elliptic curve. The **runmqakm** command can be used to request digital certificates for this security level using a -sig_alg parameter of EC_ecdsa_with_SHA256, or EC_ecdsa_with_SHA384.

At the 192-bit Suite B security level, the certificate subject's public key is required to use the NIST P-384 elliptic curve and to be signed with the NIST P-384 elliptic curve. The **runmqakm** command can be used to request digital certificates for this security level using a -sig_alg parameter of EC_ecdsa_with_SHA384.

The supported NIST elliptic curves are as follows:

Table 5. Supported NIST elliptic curves

NIST FIPS 186-3 curve name	RFC 4492 curve name	Elliptic Curve key size (bits)
P-256	secp256r1	256
P-384	secp384r1	384
P-521	secp521r1	521

Note: The NIST P-521 elliptic curve cannot be used for Suite B compliant operation.

Related concepts

"Enabling CipherSpecs" on page 411
Enable a CipherSpec by using the **SSLCIPH** parameter in either the **DEFINE CHANNEL** or **ALTER CHANNEL** MQSC command.

"Specifying that only FIPS-certified CipherSpecs are used at run time on the MQI client" on page 264
Create your key repositories using FIPS-compliant software, then specify that the channel must use FIPS-certified CipherSpecs.

"NSA Suite B Cryptography in IBM MQ" on page 42
This topic provides information about how to configure IBM MQ for AIX, Linux, and Windows to conform to the Suite B compliant TLS 1.2 profile.

"National Security Agency (NSA) Suite B Cryptography" on page 24
The government of the Unites States of America produces technical advice on IT systems and security, including data encryption. The US National Security Agency (NSA) recommends a set of interoperable cryptographic algorithms in its Suite B standard.

Channel authentication records

To exercise more precise control over the access granted to connecting systems at a channel level, you can use channel authentication records.

You might find that clients attempt to connect to your queue manager using a blank user ID or a high-level user ID that would allow the client to perform undesirable actions. You can block access to these clients using channel authentication records. Alternatively, a client might assert a user ID that is valid on the client platform but is unknown or of an invalid format on the server platform. You can use a channel authentication record to map the asserted user ID to a valid user ID.

You might find a client application that connects to your queue manager and behaves badly in some way. To protect the server from the issues this application is causing, it needs to be temporarily blocked using the IP address the client application is on until such time as the firewall rules are updated or the client application is corrected. You can use a channel authentication record to block the IP address from which the client application connects.

If you have set up an administration tool such as the IBM MQ Explorer, and a channel for that specific use, you might want to ensure that only specific client computers can use it. You can use a channel authentication record to allow the channel to be used only from certain IP addresses.

If you are just getting started with some sample applications running as clients, see Preparing and running the sample programs for an example of setting up the queue manager securely using channel authentication records.

To get channel authentication records to control inbound channels, use the MQSC command **ALTER QMGR CHLAUTH(ENABLED)**.

CHLAUTH rules are applied for a channel MCA that is created in response to a new inbound connection. For a channel MCA created in response to the channel being started locally, no **CHLAUTH** rules are applied.

Table 6. Where CHLAUTH rules are applied for different channel pairs

Channel type	**MCA where CHLAUTH rules are applied**
SDR-RCVR	RCVR
RQSTR-SVR (Started at SVR)	RQSTR
RQSTR-SVR (Started at RQSTR)	SVR
RQSTR-SDR (Started at SDR)	RQSTR
RQSTR-SDR (Started at RQSTR)	SDR for initial connection. RQSTR for callback connection.

Channel authentication records can be created to perform the following functions:

- To block connections from specific IP addresses.
- To block connections from specific user IDs.
- To set an MCAUSER value to be used for any channel connecting from a specific IP address.
- To set an MCAUSER value to be used for any channel asserting a specific user ID.
- To set an MCAUSER value to be used for any channel having a specific SSL or TLS Distinguished Name (DN).
- To set an MCAUSER value to be used for any channel connecting from a specific queue manager.
- To block connections claiming to be from a certain queue manager unless the connection is from a specific IP address.
- To block connections presenting a certain SSL or TLS certificate unless the connection is from a specific IP address.

These uses are explained further in the following sections.

You create, modify, or remove channel authentication records using the MQSC command **`SET CHLAUTH`** or the PCF command **`Set Channel Authentication Record`**.

Note: Large numbers of channel authentication records can have a negative impact on a queue manager's performance.

Blocking IP addresses

It is normally the role of a firewall to prevent access from certain IP addresses. However, there might be occasions where you experience connection attempts from an IP address that should not have access to your IBM MQ system and must temporarily block the address before the firewall can be updated. These connection attempts might not be coming from IBM MQ channels; these connection attempts might be coming from other socket applications that are misconfigured to target your IBM MQ listener. Block IP addresses by setting a channel authentication record of type BLOCKADDR. You can specify one or more single addresses, ranges of addresses, or patterns including wildcards.

Whenever an inbound connection is refused because the IP address is blocked in this manner, an event message MQRC_CHANNEL_BLOCKED with reason qualifier MQRQ_CHANNEL_BLOCKED_ADDRESS is issued, provided that channel events are enabled and the queue manager is running. Additionally, the connection is held open for 30 seconds prior to returning the error to ensure the listener is not flooded with repeated attempts to connect that are blocked.

To block IP addresses only on specific channels, or to avoid the delay before the error is reported, set a channel authentication record of type ADDRESSMAP with the USERSRC(NOACCESS) parameter.

Whenever an inbound connection is refused for this reason, an event message MQRC_CHANNEL_BLOCKED with reason qualifier MQRQ_CHANNEL_BLOCKED_NOACCESS is issued, provided that channel events are enabled and the queue manager is running.

See "Blocking specific IP addresses" on page 375 for an example.

Blocking user IDs

To prevent certain user IDs from connecting over a client channel, set a channel authentication record of type BLOCKUSER. This type of channel authentication record applies only to client channels, not to message channels. You can specify one or more individual user IDs to be blocked, but you cannot use wildcards.

Whenever an inbound connection is refused for this reason, an event message MQRC_CHANNEL_BLOCKED with reason qualifier MQRQ_CHANNEL_BLOCKED_USERID is issued, provided that channel events are enabled.

See "Blocking specific user IDs" on page 377 for an example.

You can also block any access for specified user IDs on certain channels by setting a channel authentication record of type USERMAP with the USERSRC(NOACCESS) parameter.

Whenever an inbound connection is refused for this reason, an event message MQRC_CHANNEL_BLOCKED with reason qualifier MQRQ_CHANNEL_BLOCKED_NOACCESS is issued, provided that channel events are enabled and the queue manager is running.

See "Blocking access for a client user ID" on page 380 for an example.

Blocking queue manager names

To specify that any channel connecting from a specified queue manager is to have no access, set a channel authentication record of type QMGRMAP with the USERSRC(NOACCESS) parameter. You can specify a single queue manager name or a pattern including wildcards. There is no equivalent of the BLOCKUSER function to block access from queue managers.

Whenever an inbound connection is refused for this reason, an event message MQRC_CHANNEL_BLOCKED with reason qualifier MQRQ_CHANNEL_BLOCKED_NOACCESS is issued, provided that channel events are enabled and the queue manager is running.

See "Blocking access from a remote queue manager" on page 380 for an example.

Blocking SSL or TLS DNs

To specify that any user presenting an SSL or TLS personal certificate containing a specified DN is to have no access, set a channel authentication record of type SSLPEERMAP with the USERSRC(NOACCESS) parameter. You can specify a single distinguished name or a pattern including wildcards. There is no equivalent of the BLOCKUSER function to block access for DNs.

Whenever an inbound connection is refused for this reason, an event message MQRC_CHANNEL_BLOCKED with reason qualifier MQRQ_CHANNEL_BLOCKED_NOACCESS is issued, provided that channel events are enabled and the queue manager is running.

See "Blocking access for an SSL or TLS Distinguished Name" on page 381 for an example.

Mapping IP addresses to user IDs to be used

To specify that any channel connecting from a specified IP address is to use a specific MCAUSER, set a channel authentication record of type ADDRESSMAP. You can specify a single address, a range of addresses, or a pattern including wildcards.

If you use a port forwarder, DMZ session break, or any other setup that changes the IP address presented to the queue manager, then mapping IP addresses is not necessarily suitable for your use.

See "Mapping an IP address to an MCAUSER user ID" on page 381 for an example.

Mapping queue manager names to user IDs to be used

To specify that any channel connecting from a specified queue manager is to use a specific MCAUSER, set a channel authentication record of type QMGRMAP. You can specify a single queue manager name or a pattern including wildcards.

See "Mapping a remote queue manager to an MCAUSER user ID" on page 378 for an example.

Mapping user IDs asserted by a client to user IDs to be used

To specify that if a certain user ID is used by a connection from an IBM MQ MQI client, a different, specified MCAUSER is to be used, set a channel authentication record of type USERMAP. User ID mapping does not use wildcards.

See "Mapping a client user ID to an MCAUSER user ID" on page 378 for an example.

Mapping SSL or TLS DNs to user IDs to be used

To specify that any user presenting an SSL/TLS personal certificate containing a specified DN is to use a specific MCAUSER, set a channel authentication record of type SSLPEERMAP. You can specify a single distinguished name or a pattern including wildcards.

See "Mapping an SSL or TLS Distinguished Name to an MCAUSER user ID" on page 379 for an example.

Mapping queue managers, clients, or SSL or TLS DNs according to IP address

In some circumstances it might be possible for a third party to spoof a queue manager name. An SSL or TLS certificate or key database file might also be stolen and reused. To protect against these threats, you can specify that a connection from a certain queue manager or client, or using a certain DN must be connecting from a specified IP address. Set a channel authentication record of type USERMAP, QMGRMAP or SSLPEERMAP and specify the permitted IP address, or pattern of IP addresses, using the ADDRESS parameter.

See "Mapping a remote queue manager to an MCAUSER user ID" on page 378 for an example.

Interaction between channel authentication records

It is possible that a channel attempting to make a connection matches more than one channel authentication record, and that these have contradictory effects. For example, a channel might assert a user ID which is blocked by a BLOCKUSER channel authentication record, but with an SSL or TLS certificate that matches an SSLPEERMAP record that sets a different user ID. In addition, if channel authentication records use wildcards, a single IP address, queue manager name, or SSL or TLS DN might match several patterns. For example, the IP address 192.0.2.6 matches the patterns 192.0.2.0-24, 192.0.2.*, and 192.0.*.6. The action taken is determined as follows.

- The channel authentication record used is selected as follows:
  - A channel authentication record explicitly matching the channel name takes priority over a channel authentication record matching the channel name by using a wildcard.
  - A channel authentication record using an SSL or TLS DN takes priority over a record using a user ID, queue manager name, or IP address.
  - A channel authentication record using a user ID or queue manager name takes priority over a record using an IP address.
- If a matching channel authentication record is found and it specifies an MCAUSER, this MCAUSER is assigned to the channel.
- If a matching channel authentication record is found and it specifies that the channel has no access, an MCAUSER value of *NOACCESS is assigned to the channel. This value can later be changed by a security exit program.
- If no matching channel authentication record is found, or a matching channel authentication record is found and it specifies that the user ID of the channel is to be used, the MCAUSER field is inspected.
  - If the MCAUSER field is blank, the client user ID is assigned to the channel.
  - If the MCAUSER field is not blank, it is assigned to the channel.
- Any security exit program is run. This exit program might set the channel user ID or determine that access is to be blocked.
- If the connection is blocked or the MCAUSER is set to *NOACCESS, the channel ends.
- If the connection is not blocked, for any channel except a client channel, the channel user ID determined in the previous steps is checked against the list of blocked users.
  - If the user ID is in the list of blocked users, the channel ends.
  - If the user ID is not in the list of blocked users, the channel runs.

Where a number of channel authentication records match a channel name, IP address, host name, queue manager name, or SSL or TLS DN, the most specific match is used. The match considered to be:

- The most specific is a name without wildcard characters, for example:
  - A channel name of A.B.C
  - An IP address of 192.0.2.6
  - A host name of `hursley.ibm.com`
  - A queue manager name of 192.0.2.6
- The most generic is a single asterisk (*) that matches, for example:

  - All channel names
  - All IP addresses
  - All host names
  - All queue manager names
- A pattern with an asterisk at the start of a string is more generic than a defined value at the start of a string:
  - For channels, *.B.C is more generic than A.*
  - For IP addresses, *.0.2.6 is more generic than 192.*
  - For host names, `*.ibm.com` is more generic than `hursley.*`
  - For queue manager names, *QUEUEMANAGER is more generic than QUEUEMANAGER*
- A pattern with an asterisk at a specific place in a string is more generic than a defined value at the same place in a string, and similarly for each subsequent place in a string:
  - For channels, A.*.C is more generic than A.B.*
  - For IP addresses, 192.*.2.6 is more generic than 192.0.*.
  - For host names, `hursley.*.com` is more generic than `hursley.ibm.*`
  - For queue manager names, Q*MANAGER is more generic than QUEUE*
- Where two or more patterns have an asterisk at a specific place in a string, the one with fewer nodes following the asterisk is more generic:
  - For channels, A.* is more generic than A.*.C
  - For IP addresses, 192.* is more generic than 192.*.2.*.
  - For host names, `hurlsey.*` is more generic than `hursley.*.com`
  - For queue manager names, Q* is more generic than Q*MGR
- Additionally, for an IP address:
  - A range indicated with a hyphen (-), is more specific than an asterisk. Thus 192.0.2.0-24 is more specific than 192.0.2.*.
  - A range that is a subset of another is more specific than the larger range. Thus 192.0.2.5-15 is more specific than 192.0.2.0-24.
  - Overlapping ranges are not permitted. For example, you cannot have channel authentication records for both 192.0.2.0-15 and 192.0.2.10-20.
  - A pattern cannot have fewer than the required number of parts, unless the pattern ends with a single trailing asterisk. For example 192.0.2 is invalid, but 192.0.2.* is valid.
  - A trailing asterisk must be separated from the rest of the address by the appropriate part separator (a dot (.) for IPv4, a colon (:) for IPv6). For example, 192.0* is not valid because the asterisk is not in a part of its own.
  - A pattern can contain additional asterisks, provided that no asterisk is adjacent to the trailing asterisk. For example, 192.*.2.* is valid, but 192.0.*.* is not valid.
  - An IPv6 address pattern cannot contain a double colon and a trailing asterisk, because the resulting address would be ambiguous. For example, 2001::* could expand to 2001:0000:*, 2001:0000:0000:* and so on
- For an SSL or TLS Distinguished Name (DN), the precedence order of substrings is as follows:

Table 7. Precedence order of substrings

Order	DN substring	Name
1	SERIALNUMBER=	Certificate serial number
2	MAIL=	Email address

Table 7. Precedence order of substrings (continued)

Order	DN substring	Name
3	Deprecated E=	Email address (Deprecated in preference to MAIL)
4	UID=, USERID=	User identifier
5	CN=	Common name
6	T=	Title
7	OU=	Organizational unit
8	DC=	Domain component
9	O=	Organization
10	STREET=	Street / First line of address
11	L=	Locality
12	ST=, SP=, S=	State or province name
13	PC=	Postal code / zip code
14	C=	Country
15	UNSTRUCTUREDNAME=	Host name
16	UNSTRUCTUREDADDRESS=	IP address
17	DNQ=	Distinguished name qualifier

Thus, if an SSL or TLS certificate is presented with a DN containing the substrings O=IBM and C=UK, IBM MQ uses a channel authentication record for O=IBM in preference to one for C=UK, if both are present.

A DN can contain multiple OUs, which must be specified in hierarchical order with the large organizational units specified first. If two DNs are equal in all respects except for their OU values, the more specific DN is determined as follows:

1. If they have different numbers of OU attributes then the DN with the most OU values is more specific. This is because the DN with more Organizational Units fully qualifies the DN in more detail and provides more matching criteria. Even if its top-level OU is a wildcard (OU=*), the DN with more OUs is still regarded as more specific overall.
2. If they have the same number of OU attributes then the corresponding pairs of OU values are compared in sequence left-to-right, where the left-most OU is the highest-level (least specific), according to the following rules.
   a. An OU with no wildcard values is the most specific because it can only match exactly one string.
   b. An OU with a single wildcard at either the beginning or end (for example, OU=ABC* or OU=*ABC) is next most specific.
   c. An OU with two wildcards for example, OU=*ABC*) is next most specific.
   d. An OU consisting only of an asterisk (OU=*) is the least specific.
3. If the string comparison is tied between two attribute values of the same specificity then whichever attribute string is longer is more specific.
4. If the string comparison is tied between two attribute values of the same specificity and length then the result is determined by a case-insensitive string comparison of the portion of the DN excluding any wildcards.

If two DNs are equal in all respects except for their DC values, the same matching rules apply as for OUs except that in DC values the left-most DC is the lowest-level (most specific) and the comparison ordering differs accordingly.

Displaying channel authentication records

To display channel authentication records, use the MQSC command **`DISPLAY CHLAUTH`** or the PCF command **`Inquire Channel Authentication Records`**. You can choose to return all records that match the supplied channel name, or you can choose an explicit match. The explicit match tells you which channel authentication record would be used if a channel attempted to make a connection from a specific IP address, from a specific queue manager or using a specific user ID, and, optionally, presenting an SSL/TLS personal certificate containing a specified DN.

Related concepts

"Security for remote messaging" on page 100
This section deals with remote messaging aspects of security.

Interaction of CHLAUTH and CONNAUTH

How channel authentication records (CHLAUTH) and connection authentication (CONNAUTH) interact in IBM MQ, in the case of a single conversation on a channel.

Different types of bindings

IBM MQ supports two methods for an application to connect:

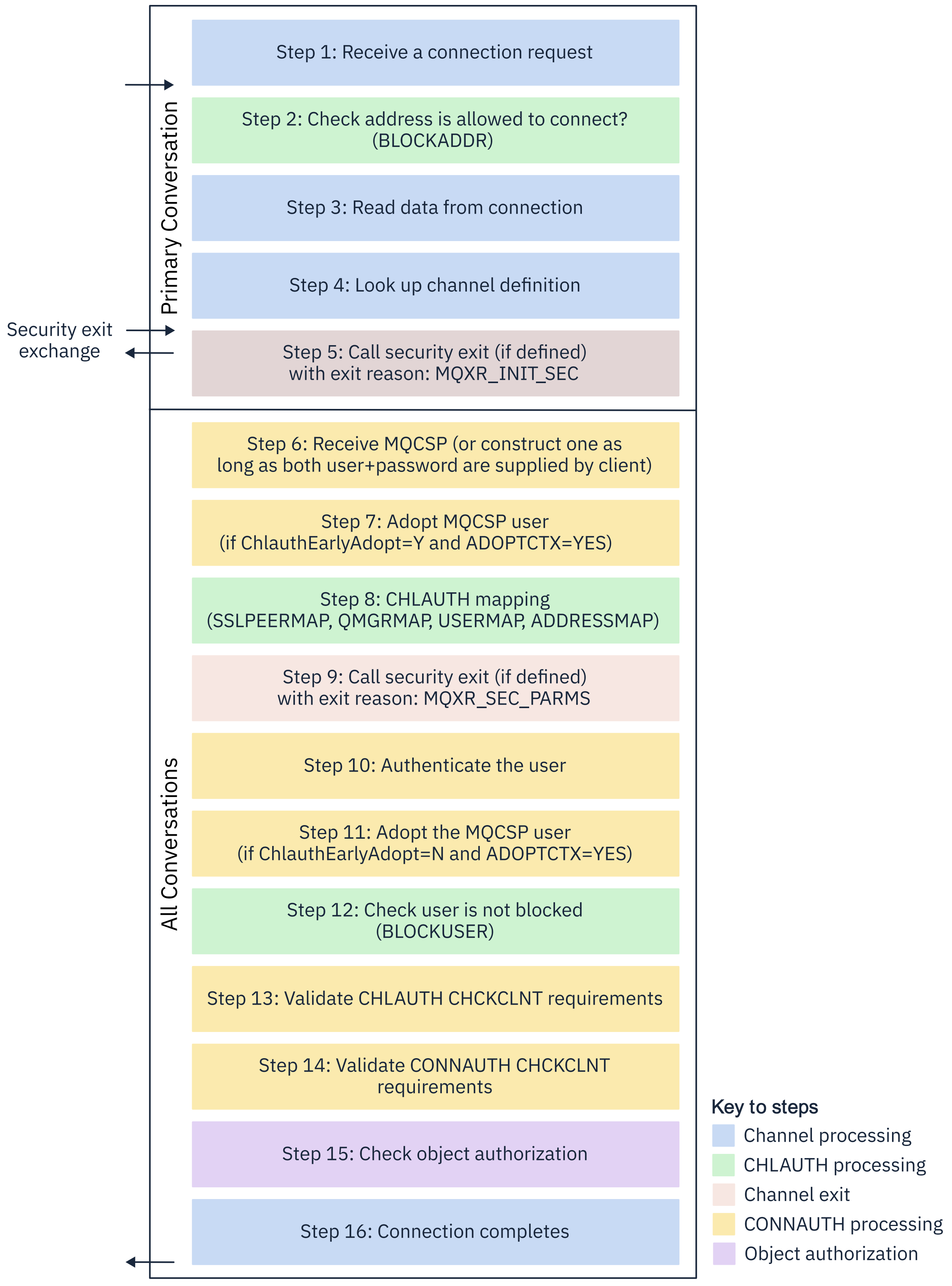
Local bindings
Applies when the application and queue manager are on the same operating image. CHLAUTH is not relevant to this type of application connection.

Client bindings
Applies when the application and queue manager use the network to communicate. The application and queue manager can be running on the same machine, or they can be on different machines. In IBM MQ, a client connection is handled in the form of a server-connection (SVRCONN) channel and, in this situation, both CONNAUTH and CHLAUTH are applicable.

Binding steps of the receiving end of a channel

When an application connects to a queue manager, a substantial amount of checking is performed to ensure that both ends of the channel understand what is supported by the other end. The receiving end of the channel does some extra checking, involving CHLAUTH and CONNAUTH, to ensure that the client is allowed to connect, and this process might also include a security exit as this can affect the result. This channel connecting phase is also referred to as the *binding phase*.

The following diagram lists the steps that a SVRCONN channel goes through when the server end (at the queue manager) starts:


Primary Conversation
Step 1: Receive a connection request
Step 2: Check address is allowed to connect? (BLOCKADDR)
Step 3: Read data from connection
Step 4: Look up channel definition
Security exit exchange
Step 5: Call security exit (if defined) with exit reason: MQXR_INIT_SEC
All Conversations
Step 6: Receive MQCSP (or construct one as long as both user+password are supplied by client)
Step 7: Adopt MQCSP user (if ChlauthEarlyAdopt=Y and ADOPTCTX=YES)
Step 8: CHLAUTH mapping (SSLPEERMAP, QMGRMAP, USERMAP, ADDRESSMAP)
Step 9: Call security exit (if defined) with exit reason: MQXR_SEC_PARMS
Step 10: Authenticate the user
Step 11: Adopt the MQCSP user (if ChlauthEarlyAdopt=N and ADOPTCTX=YES)
Step 12: Check user is not blocked (BLOCKUSER)
Step 13: Validate CHLAUTH CHCKCLNT requirements
Step 14: Validate CONNAUTH CHCKCLNT requirements
Step 15: Check object authorization
Step 16: Connection completes
Key to steps
Channel processing
CHLAUTH processing
Channel exit
CONNAUTH processing
Object authorization

Step 1: Receive a connection request
The channel initiator or listener receives a connection request from somewhere on the network.

Step 2: Is the address allowed to connect?
Before any data is read, IBM MQ checks the IP address of the partner against the CHLAUTH rules, to see if the address is in the *BLOCKADDR* rule. If the address is not found, and so not blocked, the flow proceeds to the next step.

Step 3: Read data from the channel
IBM MQ now reads the data into a buffer, and starts to process the sent information.

Step 4: Look up the channel definition
In the first data flow, IBM MQ sends, among other things, the name of the channel that the sending end is trying to start. The receiving queue manager can then look up the channel definition, which has all the settings that are specified for the channel.

Step 5: Call security exit (if defined)
If the channel has a security exit (SCYEXIT) defined, this is called with the exit reason (MQCXP.**`ExitReason`**) set to MQXR_INIT_SEC.

Step 6: Receive MQCSP
If necessary, construct one if the client supplied authentication credentials.

If the client is a Java or JMS application running in compatibility mode, the client does not pass an MQCSP structure to the queue manager. Instead, if the application supplied a user ID and password, an MQCSP structure is constructed here.

Step 7: Adopt MQCSP user (if `ChlauthEarlyAdopt` is *Y* and ADOPTCTX=YES)
The credentials that are supplied by the client are authenticated.

If CONNAUTH is using LDAP to map an asserted distinguished name to a short user ID, the mapping happens in this step.

If authentication is successful, the user ID is adopted by the channel and is used by the CHLAUTH mapping step.

Note: From IBM MQ 9.0.4 the **`ChlauthEarlyAdopt=`** *Y* parameter is automatically added to the channels stanza of the qm.ini file for new queue managers.

Step 8: CHLAUTH mapping
The CHLAUTH cache is inspected again to look for the mapping rules *SSLPEERMAP*, *USERMAP*, *QMGRMAP*, and *ADDRESSMAP*.

The rule that matches the incoming channel most specifically is used. If the rule has **USERSRC***(CHANNEL)* or *(MAP)*, the channel continues on binding.

If the CHLAUTH rules evaluate to a rule with **USERSRC***(NOACCESS)*, the application is blocked from connecting to the channel, unless the credentials are subsequently overridden with a valid credentials in Step 9.

Step 9: Call security exit (if defined)
If the channel has a security exit (SCYEXIT) defined, this is called with the exit reason (MQCXP.**`ExitReason`**) set to MQXR_SEC_PARMS.

A pointer to MQCSP will be present in the **`SecurityParms`** field of the MQCXP structure.

The MQCSP structure has pointers to the user ID (MQCSP.**`CSPUserIdPtr`**) and password (MQCSP.**`CSPPasswordPtr`**). V 9.4.0 From IBM MQ 9.3.4, the MQCSP structure also contains a pointer to the authentication token (MQCSP.**`TokenPtr`**).

It is possible to change the user ID and password, and authentication token, in the exit. The following example shows how a security exit would print the user ID and password values to an audit log:

```
if (pMQCXP -> ExitReason == MQXR_SEC_PARMS)
{
  /* It is not a good idea for security reasons to print out the user ID */
  /* and password but the following is shown for demonstration reasons */
  printf("User ID: %.*s Password: %.*s\n",
         pMQCXP -> SecurityParms -> CSPUserIdLength,
         pMQCXP -> SecurityParms -> CSPUserIdPtr,
         pMQCXP -> SecurityParms -> CSPPasswordLength,
         pMQCXP -> SecurityParms -> CSPPasswordPtr);
```

The exit can tell IBM MQ to close the channel, by returning *MQXCC_CLOSE_CHANNEL* in the MQCXP.**`Exitresponse`** field. Otherwise, channel processing continues to the connection authentication phase.

Note: If the asserted user is changed by the security exit, CHLAUTH mapping rules are not re-applied to the new user.

Step 10: Authenticate the user

The authentication phase happens if CONNAUTH is enabled on the queue manager.

To check this, issue the MQSC command 'DISPLAY QMGR CONNAUTH'.

z/OS The following example shows the output of the command **`DISPLAY QMGR CONNAUTH`** from a queue manager running on IBM MQ for z/OS.

```
 CSQM201I !MQ25 CSQMDRTC DISPLAY QMGR DETAILS
 QMNAME(MQ25)
 CONNAUTH(SYSTEM.DEFAULT.AUTHINFO.IDPWOS)
  END QMGR DETAILS
 CSQ9022I !MQ25 CSQMDRTC ' DISPLAY QMGR' NORMAL COMPLETION
```

Multi The following example shows the output of the command '**`DISPLAY QMGR CONNAUTH`**' from a queue manager running on IBM MQ for Multiplatforms.

```
     1 : DISPLAY QMGR CONNAUTH
AMQ8408: Display Queue Manager details.
   QMNAME(DEMO)
   CONNAUTH(SYSTEM.DEFAULT.AUTHINFO.IDPWOS)
```

The CONNAUTH value is the name of an **AUTHINFO** IBM MQ object.

As operating system authentication (**AUTHTYPE**(*IDPWOS*)) is valid on both IBM MQ for Multiplatforms and IBM MQ for z/OS, the examples use operating system authentication.

z/OS The following example shows the default AUTHINFO object with **AUTHTYPE**(*IDPWOS*) from a queue manager running on IBM MQ for z/OS.

```
 CSQM293I !MQ25 CSQMDRTC 1 AUTHINFO FOUND MATCHING REQUEST CRITERIA
 CSQM201I !MQ25 CSQMDRTC DISPLAY AUTHINFO DETAILS
 AUTHINFO(SYSTEM.DEFAULT.AUTHINFO.IDPWOS)
 AUTHTYPE(IDPWOS)
 QSGDISP(QMGR)
 ADOPTCTX(NO)
 CHCKCLNT(NONE)
 CHCKLOCL(OPTIONAL)
 FAILDLAY(1)
 DESCR()
 ALTDATE(2018-06-04)
 ALTTIME(10.43.04)
   END AUTHINFO DETAILS
 CSQ9022I !MQ25 CSQMDRTC ' DISPLAY AUTHINFO' NORMAL COMPLETION
```

Multi The following example shows the default AUTHINFO object with **AUTHTYPE**(*IDPWOS*) from a queue manager running on IBM MQ for Multiplatforms.

```
 1 : display authinfo(SYSTEM.DEFAULT.AUTHINFO.IDPWOS)
AMQ8566: Display authentication information details.
   AUTHINFO(SYSTEM.DEFAULT.AUTHINFO.IDPWOS)
   AUTHTYPE(IDPWOS)                     ADOPTCTX(NO)
   DESCR( )                             CHCKCLNT(REQDADM)
   CHCKLOCL(OPTIONAL)                   FAILDLAY(1)
   ALTDATE(2015-06-08)                  ALTTIME(16.35.16)
```

The AUTHINFO TYPE(IDPWOS) object has an attribute called CHCKCLNT. If the value is changed to *REQUIRED* all client applications have to supply valid credentials.

If the user was authenticated in Step 7, then another authentication check is not performed unless:

- The user ID and password, or the authentication token, in the SecurityParms field of the MQCXP structure was changed by a security exit in Step 9.
- The client application has connected with options requesting reconnectable functionality.

Step 11: Adopt the context of the MQCSP user (If `ChlauthEarlyAdopt`=*N* and ADOPTCTX=YES)

You can set the ADOPTCTX attribute, which controls whether the channel runs under MCAUSER, or the user ID the application has supplied.

If the user ID asserted in the MQCSP, or **`SecurityParms`** field of the MQXCP structure, has been successfully authenticated and **ADOPTCTX** is *YES*, then the context of the user resulting from steps 7 and 8 is adopted as the context to use for this application, unless the user ID and password, or the authentication token, in the **`SecurityParms`** field of the MQCXP structure was changed by a security exit in step 9.

This asserted user ID is the user ID that is checked for authorization to use IBM MQ resources.

For example, you do not have an MCAUSER set on the SVRCONN channel, and your client is running under '`johndoe`' on your Linux machine. Your application specifies user '`fred`' in the MQCSP, so the channel starts running with '`johndoe`' as the active MCAUSER. After the CONNAUTH check, the user '`fred`' is adopted, and the channel runs with '`fred`' as the active MCAUSER.

Step 12: Check the user is not blocked (BLOCKUSER)

If the CONNAUTH checking is successful, the CHLAUTH cache is then inspected again to check if the active MCAUSER is blocked by a *BLOCKUSER* rule. If the user is blocked, the channel ends.

Step 13: Validate CHLAUTH CHCKCLNT requirements

If the CHLAUTH rule that was selected in step 8 additionally specifies a CHCKCLNT value of REQUIRED or REQDADM then validation is done to ensure that a valid CONNAUTH userid was provided to meet the requirement.

- If CHCKCLNT(REQUIRED) is set a user must have been authenticated in step 7 or 10. Otherwise the connection is rejected.
- If CHCKCLNT(REQDADM) is set a user must have been authenticated in step 7 or 10 if this connection is determined to be privileged. Otherwise the connection is rejected.
- If CHCKCLNT(ASQMGR) is set then this step is skipped.

Notes:

1. If CHCKCLNT(REQUIRED) or CHCKCLNT(REQDADM) is set, but CONNAUTH is not enabled on the queue manager, the connection fails with a MQRC_SECURITY_ERROR (2063) return code due to the conflict in configuration.
2. The user is not re-authenticated in this step.

Step 14: Validate CONNAUTH CHCKCLNT requirements.

The authentication phase happens if CONNAUTH is enabled on the queue manager.

The CONNAUTH CHCKCLNT value is checked to determine what requirements are set for incoming connections:

- If CHCKCLNT(NONE) is set then this step is skipped
- If CHCKCLNT(OPTIONAL) is set then this step is skipped.
- If CHCKCLNT(REQUIRED) is set then a user must have been authenticated in step 7 or 10. Otherwise the connection is rejected.
- If CHCKCLNT(REQDADM) is set a user must have been authenticated in step 7 or 10 if this connection is determined to be privileged. Otherwise the connection is rejected.

Note: The user is not re-authenticated in this step.

Multi **Step 15: Check object authorization**

A check is made to ensure that the active MCAUSER has the appropriate authority to connect to the queue manager.

ALW See Object Authority Manager, for more information.

IBM i See "Object authority manager on IBM i" on page 156, for more information.

Step 16: The connection completes

If the preceding steps complete successfully, the connection completes.

Related concepts

CONNAUTH

A queue manager can be configured to authenticate credentials that are supplied by an application when it connects.

Related reference

SET CHLAUTH

ALTER AUTHINFO

Resolving CHLAUTH access issues

Steps and examples to resolve certain access issues when using channel authentication records (CHLAUTH).

Before you begin

Note: The steps in this task require you to run MQSC commands. How you do this varies by platform. See Administering IBM MQ using MQSC commands.

About this task

There are three default rules for CHLAUTH processing:

- NO ACCESS to all channels by any `MQ-admin*` users
- NO ACCESS to all SYSTEM.* channels by all users
- ALLOW access to SYSTEM.ADMIN.SVRCONN channel (non `MQ-admin` users)

The first two rules block access to all channels. The third rule is more specific, and therefore takes precedence over the other two, if the channel is the SYSTEM.ADMIN.SVRCONN channel, thus allowing access on that channel.

CHLAUTH rules are used to determine if a channel can be started, and they allow mapping, through MCAUSER to another user ID. If the channel cannot be started, the following errors commonly occur:

- `RC 2035 MQRC_NOT_AUTHORIZED`
- `RC 2059 MQRC_Q_MGR_NOT_AVAILABLE`
- `AMQ4036 Access not permitted`
- `AMQ9776: Channel was blocked by userid`
- `AMQ9777: Channel was blocked`
- `MQJE001: An MQException occurred: Completion Code 2, Reason 2035`
- `MQJE036: Queue manager rejected connection attempt`

You should block access strictly, then add more CHLAUTH rules to control who can access and start channels.

As a temporary measure, and to troubleshoot the errors listed, complete any of the following steps.

Procedure

- **Disable CHLAUTH rules**

  As a temporary measure, and also to troubleshoot the errors above, you can disable CHLAUTH rules. The rules can be re-enabled at any time, and if disabling the CHLAUTH rules resolves the connection issue, you know that this was the cause.

  To disable CHLAUTH rules run the following MQSC command:

```
ALTER QMGR CHLAUTH (DISABLED)
```

  Note that you can also set CHLAUTH to *WARN*, which allows access and logs the result of the rule.

- **Modify or remove CHLAUTH rules**

  You can also delete or modify the CHLAUTH rule, or rules, causing your problem.

  To modify a CHLAUTH rule, you use the SET CHLAUTH command with the ACTION (REPLACE). For example, to modify the default rule which causes no access to all channels by any `MQ-admin` users to WARN, instead of being blocked, run the following MQSC command:

```
SET CHLAUTH (*) TYPE (BLOCKUSER) USERLIST (*MQADMIN) WARN(YES)
ACTION (REPLACE)
```

  To delete a CHLAUTH rule, you use the SET CHLAUTH command with the ACTION (REMOVE). For example, to delete the default rule which causes no access to all channels by any `MQ-admin` users, run the following MQSC command:

```
SET CHLAUTH (*) TYPE (BLOCKUSER) USERLIST (*MQADMIN) ACTION (REMOVE)
```

- **Test access using MATCH (RUNCHECK)**

  You can test the result of your CHLAUTH rules, using the MATCH (*RUNCHECK*) option of the CHLAUTH rule. The **MATCH** (*RUNCHECK*) option returns the record that is matched by a specific inbound channel at run time, if that channel connects into this queue manager. You must provide:

  - The channel name
  - ADDRESS attribute
  - SSLPEER attribute, only if the inbound channel uses SSL or TLS
  - QMNAME, if the inbound channel is a queue manager channel, or
  - CLNTUSER attribute, if the inbound channel is a client channel

  The following example runs an MQSC command to check what CHLAUTH rule, with the default rules in place, results in an `MQ-admin` user `johndoe` accessing a channel named CHAN1:

```
DISPLAY CHLAUTH (CHAN1) MATCH (RUNCHECK) CLNTUSER ('johndoe') ADDRESS
('192.168.1.138')

AMQ8878: Display channel authentication record details.
CHLAUTH(*) TYPE(BLOCKUSER)
USERLIST(*MQADMIN)
```

  For user `johndoe`, the channel does not run, the user will be blocked due to the BLOCKUSER rule for *MQADMIN users.

  The following example runs an MQSC command to check what CHLAUTH rule, with the default rules in place, results in user `alice` who is not an `MQ-admin` user, accessing a channel named CHAN1:

```
DISPLAY CHLAUTH (CHAN1) MATCH (RUNCHECK) CLNTUSER ('alice') ADDRESS
('192.168.1.138')

AMQ9783: Channel will run using MCAUSER('alice').
```

For user `alice`, the channel runs, and the channel passes `alice` in as the MCAUSER. The MCAUSER is the user ID used to check IBM MQ object authorities.

Related reference

SET CHLAUTH

DISPLAYCHLAUTH

Creating new CHLAUTH rules for users
Some common scenarios for users, and example CHLAUTH rules to accomplish these.

Before you begin

Note: The steps in this task require you to run MQSC commands. How you do this varies by platform. See Administering IBM MQ using MQSC commands.

About this task

There are three default rules for CHLAUTH processing:

- NO ACCESS to all channels by any `MQ-admin*` users
- NO ACCESS to all SYSTEM.* channels by all users
- ALLOW access to SYSTEM.ADMIN.SVRCONN channel (non `MQ-admin` users)

The first two rules block access to all channels. The third rule is more specific, and therefore takes precedence over the other two, if the channel is the SYSTEM.ADMIN.SVRCONN channel, thus allowing access on that channel.

To create new CHLAUTH rules for users, configure one or more of the following scenarios.

Procedure

- **Control access for specific `MQ-admin` users**
  a) Set up a server connection channel that is to be exclusively used for an administrative perspective, that is, to connect from IBM MQ Explorer.

     You have a specific channel for this usage, and defined IP address, or addresses, from where you want connections to be accepted, and access blocked for the `'mqm'` ID, if the connection is not from one of the specified IP addresses.
  b) Make a SVRCONN channel for IBM MQ Explorer and `MQ-admin` users called `ADMIN.CHAN`.
     Run the following MQSC command:

```
DEFINE CHANNEL (ADMIN.CHAN) CHLTYPE (SVRCONN) TRPTYPE (TCP)
```

  c) For testing, ensure that you have a user defined that is in the `MQ-admin` group, and one that is not.

     For this scenario, `mqadm` is in the `MQ-admin` group, and `alice` is not.
  d) Confirm that the default CHLAUTH rules are in place.
  e) Add three rules to allow a specific user to access ADMIN.CHAN as `MQ-admin` from certain IP addresses:
     – Set NOACCESS from any address
     – Set BLOCKUSER for this channel to only block user `nobody`, which overrides the *MQADMIN BLOCKUSER
     – ALLOW access to user `mqadm` on a specific subnet of addresses, and MAP to `mqadm` user authority

     To do this, run the following MQSC commands:

```

SET CHLAUTH (ADMIN.CHAN) TYPE(ADDRESSMAP) ADDRESS('*') USERSRC(NOACCESS)
SET CHLAUTH('ADMIN.CHAN') TYPE(BLOCKUSER) +
DESCR('Rule to override *MQADMIN blockuser on this channel') +
USERLIST('nobody') ACTION(replace)
SET CHLAUTH('ADMIN.CHAN') TYPE(USERMAP) +
CLNTUSER('mqadm') USERSRC(MAP) MCAUSER('mqadm') +
ADDRESS('192.168.1.*') +
DESCR('Allow mqadm as mqadm on local subnet') ACTION(ADD)
```

At this point, the user `mqadm` can access and start the ADMIN.CHAN channel, from the specified IP address range.

f) Optional: You can run the MQSC command MATCH (RUNCHECK) at any time to see the results of each of these commands:

```
DISPLAY CHLAUTH (ADMIN.CHAN) MATCH (RUNCHECK) CLNTUSER ('mqadm') ADDRESS
('192.168.1.138')
AMQ8878: Display channel authentication record details.
CHLAUTH(ADMIN.CHAN) TYPE(USERMAP)
ADDRESS(192.168.1.*) CLNTUSER(mqadm)
MCAUSER(mqadm)
```

```
DISPLAY CHLAUTH (ADMIN.CHAN) MATCH (RUNCHECK) CLNTUSER ('alice') ADDRESS
('192.168.1.138')
AMQ8878: Display channel authentication record details.
CHLAUTH(ADMIN.CHAN) TYPE(ADDRESSMAP)
ADDRESS(*) USERSRC(NOACCESS)
```

At this point, only the users that have a CHLAUTH record are allowed to access using the ADMIN.CHAN.

- **Control access for a specific user and IBM MQ client application**

  For this scenario, the default CHLAUTH rules are adequate, assuming IBM MQ authority should be set for a specific user, to provide the correct IBM MQ authority (using setmqaut).

  In this scenario, the authorities are set for a user `mqapp1`, who is not an `MQ-admin` user.

  a) Use the following MQSC command to make a SVRCONN channel, APP1.CHAN, to be used by a particular application and a specific user.

```
DEFINE CHANNEL (APP1.CHAN) CHLTYPE (SVRCONN) TRPTYPE (TCP)
```

  b) With the default CHLAUTH rules in place, user `mqapp1` can start the APP1.CHAN channel.

  The user ID coming from the IBM MQ client application is used for IBM MQ object authority checking. In this case, assuming the `mqapp1` user is running the IBM MQ client app, this is used for IBM MQ object authority checking. Therefore, if `mqapp1` has access to the IBM MQ objects the application needs, all is fine; if not you will get authority errors.

  You can further increase security by creating specific CHLAUTH rules for the `mqapp1` user Id but, under the default rules, no member of the `MQ-admin` group can access this channel.

  Run the following MQSC commands:

```
SET CHLAUTH (APP1.CHAN) TYPE(ADDRESSMAP) ADDRESS('*') USERSRC(NOACCESS)
SET CHLAUTH('APP1.CHAN') TYPE(USERMAP) +
CLNTUSER('mqapp1') USERSRC(MAP) MCAUSER('mqapp1') +
DESCR('Allow mqapp1 as mqapp1 on local subnet') ACTION(ADD)
```

- **Control access for a specific user by using the certificate distinguished name (DN) of that user**

  For this scenario, the user must have a certificate that is flowed to the queue manager. The DN is then matched against the SSLPEER setting of the CHLAUTH rule, and the SSLPEER can use wildcard characters.

If matched, the user can also be mapped to a different MCAUSER for purposes of checking the IBM MQ object authorities. Mapping the MCAUSER can minimize the number of users that need to be managed in the IBM MQ object authority manager (OAM).

a) You have a TLS channel with certificates in use, and you require rules to:

   – Block all users for a particular channel
   – Allow only users with a particular SSLPEER who use the client of that user for IBM MQ OAM access.

   Run the following MQSC commands:

```
.
# block all users on any IP address.
SET CHLAUTH('SSL1.SVRCONN') TYPE(ADDRESSMAP) ADDRESS('*')
USERSRC(NOACCESS) DESCR(''block all'') WARN(NO) ACTION(ADD)
.
# override - no MQM admin rule (allow mqm group /mqm admin users to
connect.
SET CHLAUTH('SSL1.SVRCONN') TYPE(BLOCKUSER) USERLIST('nobody')
DESCR('override no mqm admin rule') WARN(NO) ACTION(ADD)
.
# allow particular SSLPEER, use client id coming in from channel
SET CHLAUTH('SSL1.SVRCONN') TYPE(SSLPEERMAP)
SSLPEER('CN=JOHNDOE,O=IBM,C=US') USERSRC(CHANNEL) ACTION(ADD)
```

   The client user ID connecting on the channel is used for the IBM MQ OAM authority of IBM MQ objects; therefore the user Id must have appropriate IBM MQ authorities.

b) Optional: Map to a different IBM MQ user ID.

   Rerun the previous MQSC command, substituting `USERSRC(MAP) MCAUSER('mquser1')` for `USERSRC(CHANNEL)`.

- **Map a particular user to the `mqm` user**

  This is an addition or modification to Control access for specific MQ-admin users.

  Use MQSC commands to add the following CHLAUTH rule to map particular users to the `mqm` user, or an `MQ-admin` user Id, that has IBM MQ object authority setup in the IBM MQ OAM.

```
SET CHLAUTH('ADMIN.CHAN') TYPE(USERMAP) +
CLNTUSER ('johndoe') USERSRC(MAP) MCAUSER ('mqm') +
ADDRESS('192.168.1-100.*') +
DESCR ('Allow johndoe as MQ-admin on local subnet') ACTION (ADD)
```

  This allows and maps the `johndoe` user over to the `mqm` user for the particular channel ADMIN.CHAN.

Related concepts

"Creating new CHLAUTH rules for channels" on page 66
To help you create your own CHLAUTH rules, here are some common scenarios for channels, and example CHLAUTH rules to accomplish these.

Related tasks

"Resolving CHLAUTH access issues" on page 62
Steps and examples to resolve certain access issues when using channel authentication records (CHLAUTH).

Related reference

SET CHLAUTH
DISPLAYCHLAUTH

Creating new CHLAUTH rules for channels
To help you create your own CHLAUTH rules, here are some common scenarios for channels, and example CHLAUTH rules to accomplish these.

This topic contains the following scenarios:

- “Only allow access to a particular channel from a specific IP address range.” on page 67
- “For a specific channel, block all users, but allow specific users to connect.” on page 67
- “Using CHLAUTH for receiver and sender channels” on page 68

Only allow access to a particular channel from a specific IP address range.

For this scenario you want to:

- Set No access to the channel from anywhere
- Allow access from a specific IP address or address range

```
runmqsc:
SET CHLAUTH('APP2.CHAN') TYPE(ADDRESSMAP) ADDRESS('*') USERSRC(NOACCESS)
WARN(NO) ACTION(ADD)
SET CHLAUTH('APP2.CHAN') TYPE(ADDRESSMAP) ADDRESS('9.95.100.1-5')
USERSRC(MAP) MCAUSER('mqapp2') ACTION(ADD)
```

This allows only the APP2.CHAN channel to be started when the connection comes from the specific IP address range specified.

The user connecting as MCAUSER is mapped to `mqapp2`, and therefore gets the IBM MQ OAM authority for that user.

For a specific channel, block all users, but allow specific users to connect.

There are three default rules for CHLAUTH processing:

- NO ACCESS to all channels by any `MQ-admin*` users
- NO ACCESS to all SYSTEM.* channels by all users
- ALLOW access to SYSTEM.ADMIN.SVRCONN channel (non `MQ-admin` users)

The first two rules block access to all channels. The third rule is more specific, and therefore takes precedence over the other two, if the channel is the SYSTEM.ADMIN.SVRCONN channel, thus allowing access on that channel.

For this scenario, the access to the channel MY.SVRCONN has the default CHLAUTH rules in place.

You need to add the following:

```
# block all users
SET CHLAUTH('MY.SVRCONN') TYPE(ADDRESSMAP) ADDRESS('*') USERSRC(NOACCESS)
DESCR(''block all'') WARN(NO) ACTION(ADD)

# override - no MQM admin rule
SET CHLAUTH('MY.SVRCONN') TYPE(BLOCKUSER) USERLIST('nobody') DESCR('override
no mqm admin rule') WARN(NO) ACTION(ADD)

# allow johndoe userid
SET CHLAUTH('MY.SVRCONN') TYPE(USERMAP) CLNTUSER('johndoe')
USERSRC(CHANNEL) DESCR('allow johndoe userid') ACTION(ADD)
```

This first part of the code blocks anyone from connecting on MY.SVRCONN, then the code allows only the MY.SVRCONN channel to be started when the connection comes from the specific user Id `johndoe`.

The user connecting on the channel `johndoe` is used for the IBM MQ OAM authority of IBM MQ objects. Therefore, the user Id must have the appropriate IBM MQ authorities.

You can map to a different IBM MQ user Id if you want to, by using:

```
USERSRC(MAP) MCAUSER('mquser1')
```

instead of `USERSRC(CHANNEL)`.

Using CHLAUTH for receiver and sender channels

You can use CHLAUTH rules to add extra security to receiver and sender channels, to restrict access to the receiver channel. Note, that if you are adding or making changes to CHLAUTH rules, the updated CHLAUTH rules only apply when starting the channel, so if the channels are already running, you need to stop and restart them, for the CHLAUTH updates to apply.

CHLAUTH rules can be used on any channel, but there are some restrictions. For example, USERMAP rules apply to SVRCONN channels only.

This example allows a connection from a particular IP address only, to start the TO.MYSVR1 channel:

```
# First you could lock down the channel by disallowing all
# for channel 'TO.MYSVR1', RCVR channel
SET CHLAUTH('TO.MYSVR1') TYPE(ADDRESSMAP) ADDRESS('*') USERSRC(NOACCESS)
DESCR('Back-stop rule')

# Then you could allow this channel to be started
SET CHLAUTH('TO.MYSVR1') TYPE(ADDRESSMAP) ADDRESS('192.168.1.134') USERSRC(MAP)
MCAUSER('mqapp') ACTION(ADD)
```

This example allows the connection from a particular queue manager only:

```
# Lock down all access:
SET CHLAUTH('TO.MYSVR1') TYPE(ADDRESSMAP) ADDRESS('*') USERSRC(NOACCESS)
DESCR('Back-stop rule')

# Then allow access from queue manager MYSVR2 and from a particular ipaddress:
SET CHLAUTH('TO.MYSVR1') TYPE(QMGRMAP) QMNAME('MYSVR2') USERSRC(MAP)
MCAUSER('mqapp') ADDRESS('192.168.1.134') ACTION(ADD)
```

Related tasks

"Resolving CHLAUTH access issues" on page 62
Steps and examples to resolve certain access issues when using channel authentication records (CHLAUTH).

"Creating new CHLAUTH rules for users" on page 64
Some common scenarios for users, and example CHLAUTH rules to accomplish these.

Related reference

SET CHLAUTH

DISPLAYCHLAUTH

Creating a CHLAUTH back-stop rule
When thinking about the control of inbound connections into your queue manager you have two options. Either you can try to list all the connections that are not allowed, or you can start by saying all connections are not allowed, and then try to list all the connections that are allowed. This second option is described here.

About this task

The reason for using the second option is, that if you try to list all the connections that are not allowed, and everything not listed is therefore allowed in, the result of missing one off the list is that a connection that should not have been allowed is able to connect, causing a potential security breach.

Conversely, if instead, you start by saying every connection is not allowed, and then list those that are, the result of missing one off this list is not a security breach. If your enterprise requires additional connections to be added, this is a relatively simple task, but there is no potential security breach.

The first thing to do is create a *back-stop* rule, which is a rule that catches any connections not otherwise matched by more specific rules. This rule has the effect of stopping any remote connections from being able to attach to your queue manager at all.

However, if you are concerned about this approach, you can set up the *back-stop* rule in warning mode; see step “2” on page 69

Procedure

1. To create a back-stop rule that stops remote connections attaching to your queue`manager, issue the following command:

   ```
   SET CHLAUTH('*') TYPE(ADDRESSMAP) ADDRESS('*')
   USERSRC(NOACCESS) DESCR('Back-stop rule')
   ```

   Now that you have closed the door on all remote connections, you can start to put more specific rules in place to allow certain connections in. For example:

   ```
   SET CHLAUTH('APPL1.SVRCONN') TYPE(ADDRESSMAP) ADDRESS('9.20.1-3.*') USERSRC(CHANNEL)
   SET CHLAUTH('SYSTEM.ADMIN.*') TYPE(SSLPEERMAP) SSLPEER('O=IBM') USERSRC(CHANNEL)
   SET CHLAUTH('TO.QM2') TYPE(QMGRMAP) QMNAME('QM1') USERSRC(MAP) MCAUSER('QM1USER')
   SET CHLAUTH('*.SVRCONN') TYPE(USERMAP) CLNTUSER('johndoe') MCAUSER('johndoe@yourdomain')
   SET CHLAUTH('*') TYPE(SSLPEERMAP) SSLPEER('CN="John Doe"') ADDRESS('9.*') MCAUSER('johndoe')
   ```

2. If you want to create the back-stop rule in warning mode, issue the following command:

   ```
   SET CHLAUTH('*') TYPE(ADDRESSMAP) ADDRESS('*')
   USERSRC(NOACCESS) DESCR('Back-stop rule') WARN(YES)
   ```

   Now you can continue, and make all your positive rules. When you believe you have created all the rules you need, turn on channel events by issuing the following command:

   ```
   ALTER QMGR CHLEV(EXCEPTION)
   ```

   and monitor the SYSTEM.ADMIN.CHANNEL.EVENT queue for events with **Reason** set to MQRC_CHANNEL_BLOCKED_WARNING.

   These events detail the connections that have matched your back-stop rule, but because the command is running in warning mode, have not actually been blocked for the moment.

   Review each of these events and determine whether this connection should have a positive rule in place to allow it in, or whether it has correctly been matched against the *back-stop* rule. You can run in this mode, reviewing the events as they are created, until you are happy that you have seen all the inbound channels, and have appropriate positive rules in place for them all.

   At this point, you can change the *back-stop* rule to start really blocking connections that it matches by issuing the following command:

   ```
   SET CHLAUTH('*') TYPE(ADDRESSMAP) ADDRESS('*')
   USERSRC(NOACCESS) DESCR('Back-stop rule') WARN(NO)
   ACTION(REPLACE)
   ```

Creating a non-privileged IBM MQ administrator
How you create a non-privileged IBM MQ administrator using CHLAUTH.

About this task

In the context of this task, the terms:

privileged user
Means a user that has authorization to perform an operation without being explicitly granted access to do that operation. The users in the mqm group are examples of these privileged users.

IBM MQ administrator
Means a user who has a need to issue administrative commands against IBM MQ, such as **`DEFINE QLOCAL`** or **`START CHANNEL`**.

The following steps create a non-privileged IBM MQ administrator.

Procedure

1. Create a user ID on the queue manager machine using the appropriate commands for the platform, or platforms, your enterprise uses.

   The user name `alice` is used in this example.
2. Grant this new user authority to issue all IBM MQ administrative commands by carrying out the following procedure:

   a) Start up the IBM MQ Explorer using a privileged user.

   b) Navigate to the *Role Based Wizard* by selecting the appropriate queue manager, then `Object Authorities` and `Add Role Based Authorities`.

   c) In the wizard panel that pops up, enter the user ID you created in the first step, or if you prefer to work with groups, enter the group name for the user or set of users that you want to make into non-privileged IBM MQ administrators.

   d) Set up the wizard for full administrative access.

   e) If you want to allow your non-privileged IBM MQ administrator to be able to browse messages on queues, also select that check box.

   f) Review the commands in the preview panel at the bottom of the wizard.

   You can cut and paste these commands to build your own scripts.

   One reason you might prefer to do this with your own script is to reduce the amount of access you give to this user. Perhaps rather than granting access to all objects, you might prefer to only grant access to a certain group of objects.

   Pressing **OK** on the wizard issues the commands as they are shown.

   g) You need to set up some CHLAUTH rules to allow remote access for this user ID, if the requirement for a non-privileged IBM MQ administrator is to be for remote access as well.

   Assuming that your enterprise is using the guidance in "Creating a CHLAUTH back-stop rule" on page 68, all you need to do is add an enabling rule.

   The rule you create rather depends on how you choose to authenticate your remote IBM MQ administrators.

   If you are using weak TCP/IP authentication, you might set up a CHLAUTH rule which looks like the following:

   ```
     SET CHLAUTH(admin-channel-name)      TYPE(ADDRESSMAP)
   ADDRESS('1.2.3.4')      USERSRC(MAP) MCAUSER('alice')
   DESCR('Admin Channel - Weak TCP/IP authentication')
   ```

   9. If you are using TLS authentication, you might set up a CHLAUTH rule which looks like the following:

   ```
   SET CHLAUTH(admin-channel-name)      TYPE(SSLPEERMAP)
   SSLPEER('CN=Alice') ADDRESS('1.2.3.4')      USERSRC(MAP) MCAUSER('alice')
   DESCR('Admin Channel - TLS authentication'
   ```

   Now, when a user connects into the `admin-channel-name` (and matches the CHLAUTH rules) they are able to issue commands under the user ID `alice` on the queue manager, and so privileged remote access is not required.

Connection authentication

Connection authentication allows applications to supply authentication credentials when they connect to a queue manager. The queue manager validates the credentials. The user ID supplied in the credentials can also be adopted for use in authorization checks for resources that the application accesses.

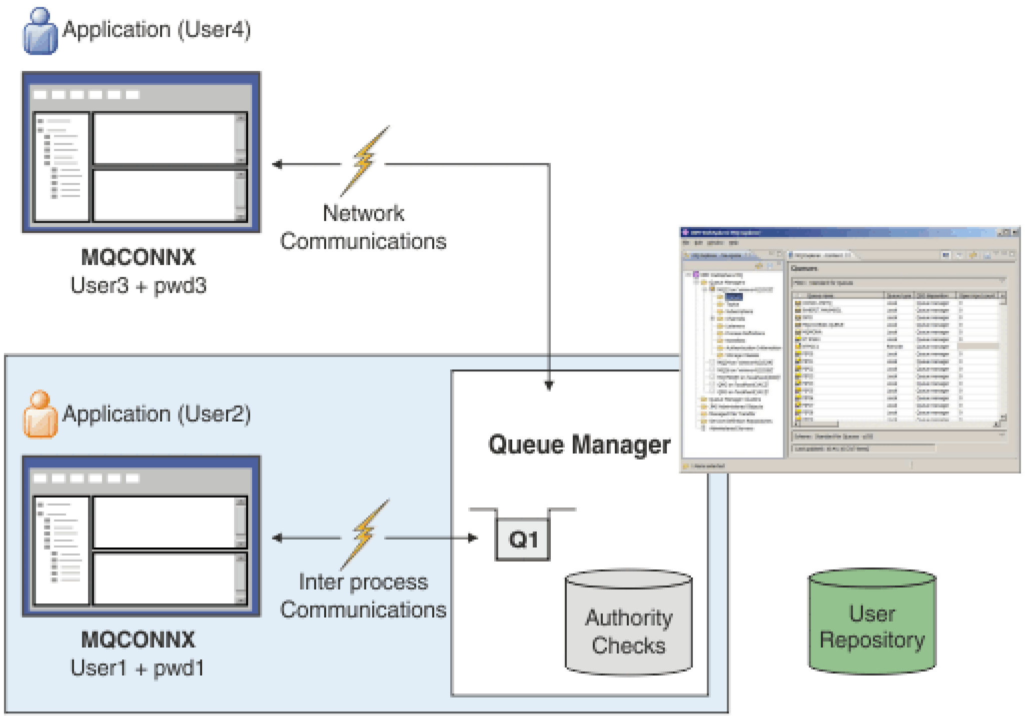
Applications can supply a user ID and password for authentication when they connect to a queue manager.

V 9.4.0 From IBM MQ 9.3.4, IBM MQ client applications can also supply an authentication token as an alternative method of authentication.

The queue manager can be configured to validate the credentials that are supplied by the application.

A user ID and password that is supplied by an application is checked by using the user repository in the queue manager configuration. For more information about the repository that is used for checking user IDs and passwords, see User repositories.

V 9.4.0 Authentication tokens are validated by using the certificates and symmetric keys in the queue manager's token authentication keystore to validate the token's signature. For more information about authenticating users with authentication tokens, see "Working with authentication tokens" on page 318.



In the diagram, two applications are making connections with a queue manager, one application as a client and one using local bindings. Applications might use various APIs to connect to the queue manager, but all have the ability to provide a user ID and a password. The user ID that the application is running under, `User2` and `User4` in the diagram, which is the usual operating system user ID presented to IBM MQ, might be different from the user ID provided by the application, `User1` and `User3`.

The queue manager receives configuration commands (in the diagram, IBM MQ Explorer is being used) and manages the opening of resources and checks the authority to access those resources. There are many different resources in IBM MQ that an application might require authority to access. The diagram illustrates opening a queue for output, but the same principles apply to other resources as well.

Related concepts

"Connection authentication: Configuration" on page 72
A queue manager can be configured to authenticate credentials that are supplied by an application when it connects.

"Connection authentication: Application changes" on page 76

"Connection authentication: User repositories" on page 78

For each of your queue managers, you can choose different types of authentication information object for authenticating user IDs and passwords.

Connection authentication: Configuration

A queue manager can be configured to authenticate credentials that are supplied by an application when it connects.

Turning on connection authentication on a queue manager

On a queue manager object, the **CONNAUTH** attribute can be set to the name of an authentication information (AUTHINFO) object. The **AUTHTYPE** attribute of an AUTHINFO object specifies the type of the object. AUTHINFO objects that are used for connection authentication can be one of the following two types:

IDPWOS
: The queue manager uses the local operating system to authenticate the user ID and password that is supplied by a connecting application.

  Linux V 9.4.0 AIX From IBM MQ 9.3.4, this type of AUTHINFO object also allows a queue manager that runs on AIX or Linux to validate authentication tokens. In addition to the AUTHINFO object that is used to configure connection authentication, the queue manager must be configured to accept authentication tokens with the **AuthInfo** stanza of the qm.ini file. For more information about configuring a queue manager to accept authentication tokens, see "Configuring a queue manager to accept authentication tokens using a local keystore" on page 325.

IDPWLDAP
: The queue manager uses an LDAP server to authenticate the user ID and password that is supplied by a connecting application.

Note: You cannot specify any other type of authentication information object in the queue manager's **CONNAUTH** attribute.

AUTHINFO objects of type IDPWOS and IDPWLDAP are similar in several of their attributes. The attributes described here are common to both types of objects.

The following example MQSC commands turn on connection authentication with the following operations:

1. Define an AUTHINFO object named USE.PW.
2. Alter the queue manager **CONNAUTH** attribute to refer to this AUTHINFO object.
3. Issue the **REFRESH SECURITY** command to refresh the queue manager's connection authentication configuration. The **REFRESH SECURITY** command must be issued before the queue manager recognizes any changes to the connection authentication configuration.

```
DEFINE AUTHINFO(USE.PW) +
AUTHTYPE(IDPWOS) +
FAILDLAY(10) +
CHCKLOCL(OPTIONAL) +
CHCKCLNT(REQUIRED)

ALTER QMGR CONNAUTH(USE.PW)

REFRESH SECURITY TYPE(CONNAUTH)
```

To control whether credentials are checked for connections that are made by locally bound applications, use the AUTHINFO attribute **CHCKLOCL** (check local connections). To control whether credentials are checked for connections that are made by client applications, use the AUTHINFO attribute **CHCKCLNT** (check client connections).

CHCKLOCL accepts the values of NONE and OPTIONAL, and **CHCKCLNT** allows the value of NONE for the authentication requirements to be configured:

NONE
: Authentication credentials that are supplied by applications are not checked.

`OPTIONAL`

Ensures that any credentials that are provided by an application are valid. However, it is not mandatory for applications to provide authentication credentials. This option might be useful during migration, for example.

If you:

- Provide the username and password, they are authenticated.
- Do not provide the username and password, connection is allowed.
- Provide the username, but not the password you receive an error.

Important: `OPTIONAL` is the minimum value that you can set if you also want to set a more restrictive option in channel authentication (CHLAUTH) rules.

If you select `NONE` and the client connection matches a CHLAUTH record with **`CHCKCLNT`** set to `REQUIRED` (or `REQDADM` on platforms other than z/OS), the connection fails. You receive message AMQ9793 on Multiplatforms, and message CSQX793E on z/OS.

For more information about using channel authentication rules to set more restrictive **`CHCKCLNT`** options for some client connections, see "Configuration granularity" on page 73.

`REQUIRED`

Requires that all applications provide valid credentials. See also the following note.

`REQDADM`

Privileged users must supply valid credentials, but non-privileged users are treated as with the `OPTIONAL` setting. See also the following note. ▶ z/OS (This setting is not allowed on z/OS systems.)

Note:

Setting **`CHCKLOCL`** to `REQUIRED` or `REQDADM` means that you cannot locally administer the queue manager by using **`runmqsc`** (error AMQ8135: Not authorized) unless the user specifies the **`-u`** parameter to specify the user ID in the **`runmqsc`** command. With that parameter set, **`runmqsc`** prompts for the user's password at the console.

Similarly, a user that runs IBM MQ Explorer on the local system will see error AMQ4036 when attempting to connect to the queue manager. To specify a user ID and password, right-click the local queue manager object and select **Connection Details** > **Properties...** from the menu. In the **Userid** section, enter the user ID and password to be used, then click **OK**.

Similar considerations apply to remote connections with **`CHCKCLNT`**.

The queue manager **`CONNAUTH`** attribute is blank for queue managers that are migrated from versions earlier than IBM MQ 8.0, but is set to *SYSTEM.DEFAULT.AUTHINFO.IDPWOS* for newly created queue managers. This default **`AUTHINFO`** definition has **`CHCKCLNT`** set to `REQDADM` by default.

Therefore, any existing clients that use a privileged user ID to connect must provide valid credentials.

Warning: The credentials in an MQCSP structure for a client application are sometimes sent across the network in plain text. To ensure that client credentials are protected, see "MQCSP password protection" on page 31.

Configuration granularity

The AUTHINFO object's **`CHCKLOCL`** and **`CHCKCLNT`** attributes set authentication requirements for all connections to the queue manager. In addition to these attributes, the **`CHCKCLNT`** attribute on channel authentication (CHLAUTH) rules allow more stringent authentication requirements to be set for specific client connections that match the CHLAUTH rule.

You can set the overall **`CHCKCLNT`** value to `OPTIONAL`, for example, on the AUTHINFO object, and then upgrade it to be more stringent for certain channels by setting **`CHCKCLNT`** to `REQUIRED` or `REQDADM` on the CHLAUTH rule. By default, CHLAUTH rules are defined with **`CHCKCLNT(ASQMGR)`**, so this granularity does not have to be used. For example, these MQSC commands define one CHLAUTH rule that overrides the AUTHINFO object's **CHCKCLNT** attribute, and one CHLAUTH rule that does not:

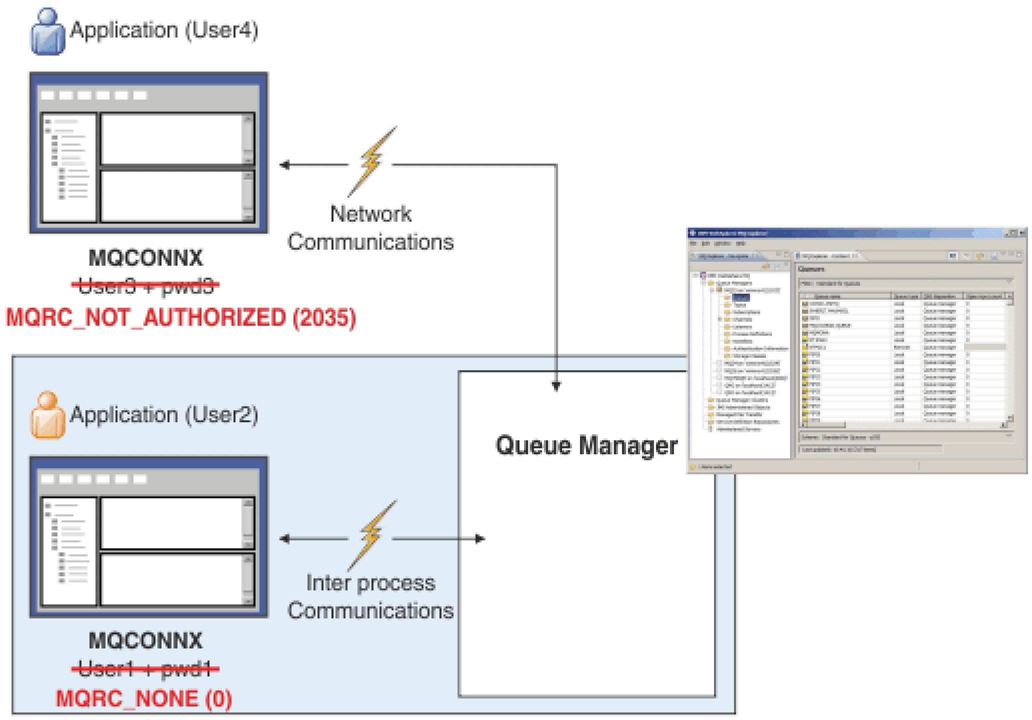
```
DEFINE AUTHINFO(USE.PW) AUTHTYPE(xxxxxx) +
CHCKCLNT(OPTIONAL)

SET CHLAUTH('*') TYPE(ADDRESSMAP) +
ADDRESS('*') USERSRC(CHANNEL) +
CHCKCLNT(REQUIRED)

SET CHLAUTH('*') TYPE(SSLPEERMAP) +
SSLPEER('CN=*') USERSRC(CHANNEL)
```

For more information about CHLAUTH rules, see "Channel authentication records" on page 51.

Error notification



An error is recorded in the following situations:

- An application does not supply authentication credentials when they are required.
- An application supplies invalid authentication credentials. This situation is treated as an error even if the configuration states that it is optional for applications to supply credentials.

Note: When **CHCKLOCL** or **CHCKCLNT** is set to NONE, invalid credentials that are supplied by applications are not detected.

Failed authentications are held for the number of seconds specified by the **FAILDLAY** attribute before the error is returned to the application. This delay provides some protection from an application repeatedly trying to connect.

The error is recorded in several ways:

Application
An MQRC_NOT_AUTHORIZED (2035) reason code is returned to the application.

Administrator
An IBM MQ administrator sees the event reported in the error log. The error message shows that the connection is rejected because the credentials are invalid, rather than because, for example, the user does not have connection authority.

Monitoring tool

A monitoring tool can also be notified of the failure, if you turn on authority events, by an event message on the SYSTEM.ADMIN.QMGR.EVENT queue. To turn on authority events, issue the following MQSC command:

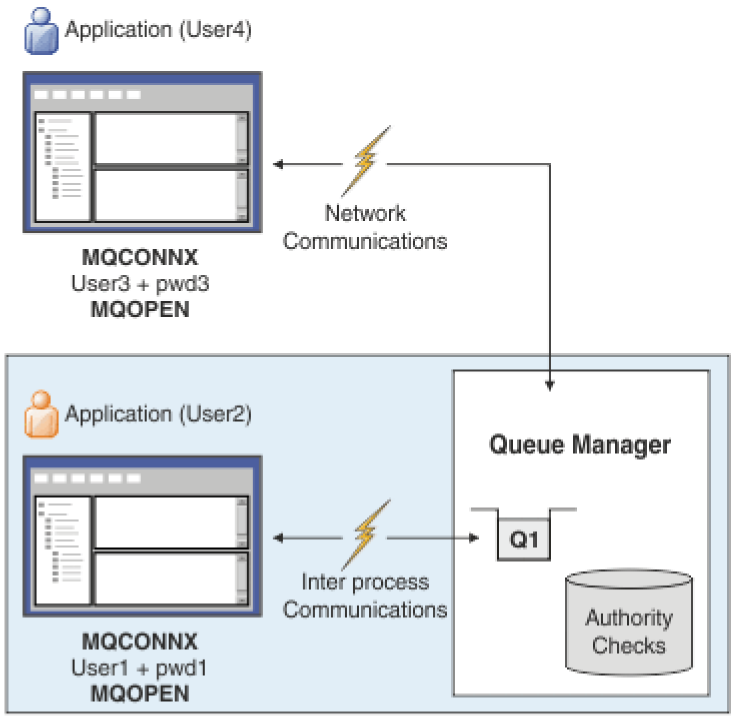
```
ALTER QMGR AUTHOREV(ENABLED)
```

This "Not Authorized" event is a Type 1 connect event, and provides the same fields as other Type 1 events, with an extra field, the MQCSP user ID that was provided. If the application supplied a password, it is not included in the event message. This means that there are two user IDs in the event message:

- The user ID that the application is running under.
- The user ID in the credentials that the application presented.

For more information about this event message, see Not Authorized (type 1).

Adopting users for authorization



You can configure the queue manager to adopt the credentials that are presented by the application as the context for the connection. Adopting the credentials means that the user ID supplied in the authentication credentials is used for authorization checks, shown on administrative displays, and appears in messages. The **ADOPTCTX** attribute on the AUTHINFO object controls whether credentials are adopted as the context for the application. For example, the following MQSC commands define an AUTHINFO object that is named USE.PWD that is used for connection authentication, and set the **ADOPTCTX** attribute to YES:

```
DEFINE AUTHINFO(USE.PWD) +
AUTHTYPE(xxxxxx) +
CHCKLOCL(OPTIONAL) +
CHCKCLNT(REQUIRED) +
ADOPTCTX(YES)

ALTER QMGR CONNAUTH(USE.PWD)
```

The following values can be specified for the **ADOPTCTX** attribute:

ADOPTCTX(YES)

The credentials that are supplied by the application are adopted as the application context for the duration of the connection. All authorization checks for an application are made with the user ID in the credentials that were authenticated.

⚠ **Attention:** When using **`ADOPTCTX(YES)`** and local operating system user IDs, you must ensure that the user ID being adopted meets the requirements for user IDs in IBM MQ. For more information, see "User IDs" on page 88.

ADOPTCTX(NO)

Credentials that are supplied by an application are used only for authentication at connection time. The user ID that the application is running under continues to be used for future authorization checks. You might find this option useful when migrating, or if you plan to use other mechanisms, such as channel authentication records, to assign the message channel agent user identifier (MCAUSER).

Interaction with Channel Authentication

Channel authentication rules can be used to change the user ID that is used as the context for an application connection, based on the user ID received from the client. For an example of using a channel authentication rule to change the user ID associated with a connection, see "Mapping a client user ID to an MCAUSER user ID" on page 378.

The order in which connection authentication and channel authentication rules are processed is a significant factor in determining the security context for IBM MQ client application connections. The **`ChlauthEarlyAdopt`** parameter in the **`channels`** stanza of the `qm.ini` file controls the order in which the queue manager adopts the context from credentials that are supplied by the application, and applies channel authentication rules. For more information about **`ChlauthEarlyAdopt`**, see Attributes of the channels stanza.

⚠ **Attention:** When you use the **`ADOPTCTX(YES)`** parameter on the authentication information object, the context that is adopted from the credentials that are supplied by the application can be changed by channel authentication rules only if the **`ChlauthEarlyAdopt`** parameter is set to Y.

For more information about the interaction of connection authentication and channel authentication, and the order in which checks take place when a client application connects to a queue manager, see "Interaction of CHLAUTH and CONNAUTH" on page 57.

Related concepts

"Connection authentication" on page 70
Connection authentication allows applications to supply authentication credentials when they connect to a queue manager. The queue manager validates the credentials. The user ID supplied in the credentials can also be adopted for use in authorization checks for resources that the application accesses.

"Connection authentication: Application changes" on page 76

"Connection authentication: User repositories" on page 78
For each of your queue managers, you can choose different types of authentication information object for authenticating user IDs and passwords.

Connection authentication: Application changes

An application that uses the message queue interface (MQI) can provide a user ID and password in the connection security parameters (MQCSP) structure when MQCONNX is called. In other application programming interfaces, the MQCSP structure is typically constructed on behalf of the application by the IBM MQ libraries.

V 9.4.0 From IBM MQ 9.3.4, client applications that connect to a queue manager that runs on AIX or Linux systems can also send an authentication token in the MQCSP structure as an alternative means of identification.

The user ID and password, or authentication token, are passed for checking to the object authority manager (OAM) supplied with the queue manager, or the authorization service component supplied with the queue manager on z/OS systems. You do not have to write your own custom interface.

If the application is running as a client, the user ID and password, or authentication token, is also passed to the client-side and server-side security exits for processing. They can also be used to set the message channel agent user identifier (MCAUSER) attribute of a channel instance.

Warning: The credentials in an MQCSP structure for a client application are sometimes sent across the network in plain text. To ensure that client application credentials are protected, see "MQCSP password protection" on page 31.

By using the XAOPEN string to provide a user ID and password, you can avoid having to change the application code.

Note:

From IBM WebSphere MQ 6.0, the security exit allows the MQCSP to be set. Therefore, clients at this level or later do not have to be upgraded.

However, in versions of IBM MQ prior to IBM MQ 8.0, MQCSP placed no restrictions on the user ID and password that were provided by the application. When using these values with features provided by IBM MQ there are limits which apply to the use of these features, but if you are only passing them to your own exits, those limits do not apply.

Related concepts

"Connection authentication" on page 70
Connection authentication allows applications to supply authentication credentials when they connect to a queue manager. The queue manager validates the credentials. The user ID supplied in the credentials can also be adopted for use in authorization checks for resources that the application accesses.

"Connection authentication: Configuration" on page 72
A queue manager can be configured to authenticate credentials that are supplied by an application when it connects.

"Connection authentication: User repositories" on page 78

For each of your queue managers, you can choose different types of authentication information object for authenticating user IDs and passwords.

Connection authentication: User repositories

For each of your queue managers, you can choose different types of authentication information object for authenticating user IDs and passwords.

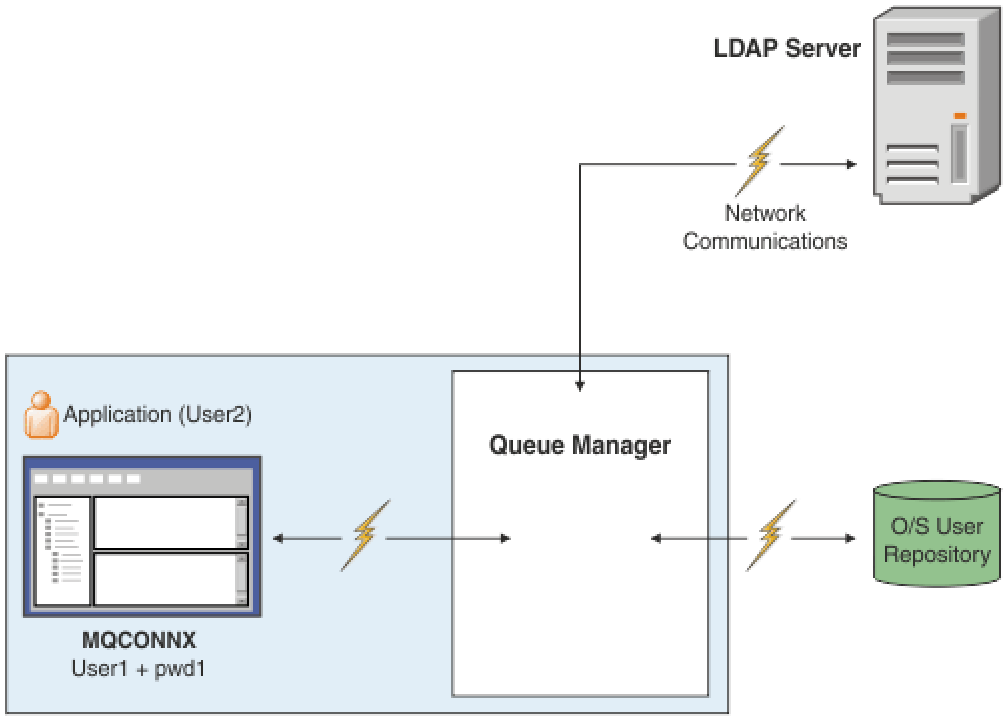


Figure 7. Types of authentication information objects

```
DEFINE AUTHINFO(USE.OS) AUTHTYPE(IDPWOS)
DEFINE AUTHINFO(USE.LDAP) +
AUTHTYPE(IDPWLDAP) +
CONNAME('ldap1(389),ldap2(389)') +
LDAPUSER('CN=QMGR1') +
LDAPPWD('passw0rd') SECCOMM(YES)
```

There are two types of authentication information object, as represented in the diagram:

- IDPWOS is used to indicate that the queue manager uses the local operating system to authentication the user ID and password. If you choose to use the local operating system, you need to set the common attributes, as described in the preceding topics.
- IDPWLDAP is used to indicate that the queue manager uses an LDAP server to authenticate the user ID and password. If you choose to use an LDAP server, more information is provided in this topic.

Only one type of authentication information object can be chosen for each queue manager to use, by naming the appropriate object in the queue manager's **CONNAUTH** attribute.

Using an LDAP server for authentication.

Set the **CONNAME** field to the address of the LDAP server for the queue manager. You can provide more addresses for the LDAP server in a comma-separated list, which can help with redundancy if the LDAP server does not provide this facility itself.

Set the required LDAP server ID and password in the **LDAPUSER** and **LDAPPWD** fields so that the queue manager can access the LDAP server and look up information about user records.

Secure connection to an LDAP Server

Unlike channels, there is no **SSLCIPH** parameter to turn on the use of TLS for communication with the LDAP server. In this case IBM MQ is acting as a client to the LDAP server so much of the configuration is done at the LDAP server. Some existing parameters in IBM MQ are used to configure how that connection works.

Set the **SECCOMM** field to control whether connectivity to the LDAP server uses TLS.

In addition to this attribute, the queue manager attributes **SSLFIPS** and **SUITEB** restrict the set of cipher specs that are chosen. The certificate that is used to identify the queue manager to the LDAP server is the queue manager certificate, either `ibmwebspheremq` *`qmgr-name`* or the value of the **CERTLABL** attribute. See Digital certificate labels for details.

LDAP User Repository

When using an LDAP user repository, there is some more configuration to be done on the queue manager other than just to tell the queue manager where to find the LDAP server.

User IDs defined in an LDAP server have a hierarchical structure that uniquely identifies them. Therefore, an application can connect to the queue manager and present its user ID as the fully qualified hierarchical user ID.

However, to simplify the information that an application must provide, it is possible to configure the queue manager to assume that the first part of the hierarchy is common to all IDs, and to automatically add this before the shortened ID provided by the application. The queue manager can then present a complete ID to the LDAP server.

Set BASEDNU to the initial point that the LDAP search looks for the ID in the LDAP hierarchy. When you set BASEDNU, you must ensure that only one result is returned when you search for the ID in the LDAP hierarchy.

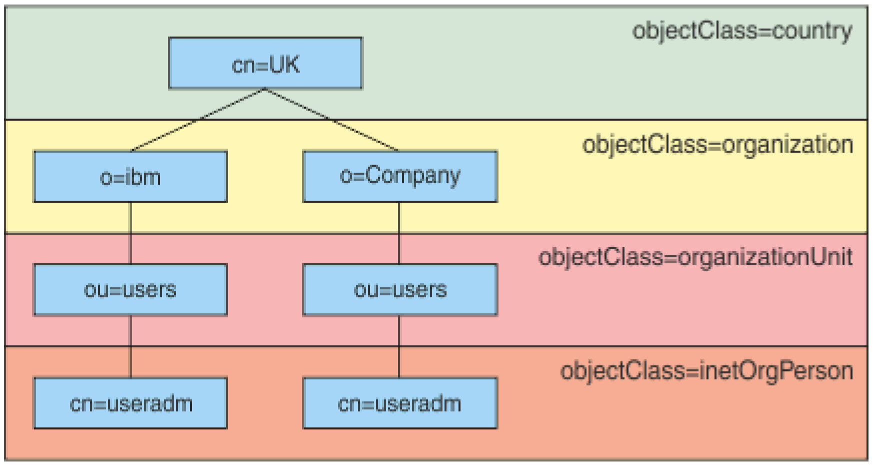


Figure 8. An example LDAP hierarchy

For example, in Figure 8 on page 79 BASEDNU can be set to "ou=users,o=ibm,c=UK" or ",o=ibm,c=UK". However, because a distinguished name that contains "cn=useradm" exists in both the "o=ibm" branch and the "o=Company" branch, BASEDNU cannot be set to "c=UK". For performance and security reasons, use the highest point in your LDAP hierarchy that from which you can reference all of the userids that you need. In this example, that is "ou=users,o=ibm,c=UK".

Your application might submit to the queue manager the user ID without providing the LDAP attribute name, CN= for example. If you set USRFIELD to the LDAP attribute name, this value is added as a prefix to the user ID that comes from the application. This might be a useful migratory aid when you move from operating system user IDs to LDAP user IDs, as the application can then present the same string in both cases and you can avoid changing the application.

Therefore, the complete user ID presented to the LDAP server looks like this:

```
USRFIELD = ID_from_application BASEDNU
```

Related concepts

"Connection authentication" on page 70
Connection authentication allows applications to supply authentication credentials when they connect to a queue manager. The queue manager validates the credentials. The user ID supplied in the credentials can also be adopted for use in authorization checks for resources that the application accesses.

"Connection authentication: Configuration" on page 72
A queue manager can be configured to authenticate credentials that are supplied by an application when it connects.

"Connection authentication: Application changes" on page 76

Client side security exit to insert user ID and password (`mqccred`)

If you have any client applications that are required to send a user ID or password but you are unable to change the source yet, there is a security exit shipped with IBM MQ 8.0 called **`mqccred`** that you can use. **`mqccred`** provides a user ID and password on behalf of the client application, from a `.ini` file. This user ID and password are sent to the queue manager which, if configured to do so, will authenticate them.

Overview

`mqccred` is a security exit that runs on the same machine as your client application. It allows user ID and password information to be supplied on behalf of the client application, where that information is not being supplied by the application itself. The user ID and password information is supplied in a structure known as the Connection Security Parameters (MQCSP) and will be authenticated by the queue manager if connection authentication is configured.

User ID and password information is retrieved from a `.ini` file on the client machine. The passwords in the file are protected by obfuscation using the **`runmqccred`** command, and also by ensuring the file permissions on the `.ini` file are set such that only the user ID running the client application (and therefore the exit) are able to read it.

Location

`mqccred` is installed:

Windows platforms
In the *`installation_directory`*`\Tools\c\Samples\mqccred\` directory

AIX and Linux platforms
In the *`installation_directory`*`/samp/mqccred` directory

Notes: The exit:

1. Acts purely as a security channel exit, and needs to be the only such exit defined on a channel.
2. Is usually named through the Client Channel Definition Table (CCDT), but a Java client can have the exit mentioned in the JNDI objects directly, or the exit might be configured for applications that manually construct the MQCD structure.
3. You must copy the **`mqccred`** and **`mqccred_r`** programs to the `var/mqm/exits` directory.

   For example, on a 64 bit AIX or Linux system, issue the command:

   ```
   cp installation_directory/samp/mqccred/lib64/* /var/mqm/exits
   ```

   See A step by step example of how to test mqccred for more information.
4. Is capable of running on previous versions of IBM MQ, as far back as IBM WebSphere MQ 7.0.1.

Setting up user IDs and passwords

The `.ini` file contains stanzas for each queue manager, with a global setting for unspecified queue managers. Each stanza contains the name of the queue manager, a user ID, and either a plain text or obfuscated password.

You must edit the `.ini` file by hand, using whichever editor you want, and add the plain text password attribute to the stanzas. Run the provided, **`runmqccred`** program, which takes the `.ini` file and replaces the **`Password`** attribute with the **`OPW`** attribute, an obfuscated form of the password.

See runmqccred for a description of the command and its parameters.

The `mqccred.ini` file contains your user ID and password information.

A template `.ini` file is provided in the same directory as the exit to provide a starting point for your enterprise.

By default, this file will be looked for in `$HOME/.mqs/mqccred.ini`. If you would like to locate it elsewhere, you can use the environment variable *MQCCRED* to point at it:

```
MQCCRED=C:\mydir\mqccred.ini
```

If you use *MQCCRED*, the variable must include the full name of the configuration file, including any `.ini` filetype. Since this file contains passwords (even if obfuscated), you are expected to protect the file using operating system privileges to ensure unauthorized people cannot read it. If you do not have the correct file permission, the exit will not run successfully.

If the application has already supplied an MQCSP structure, the exit normally respects this and will not insert any information from the `.ini` file. However, you can override this by using the **`Force`** attribute in the stanza.

Setting **`Force`** to the value *TRUE* removes the application-supplied user ID and password, and replaces those with the ini file version.

You can also set the **`Force`** attribute in the global section of the file to set the default value of that file.

The default value for **`Force`** is *FALSE*.

You can provide a user ID and password for all queue managers, or for each individual queue manager. This is an example of an `mqccred.ini` file:

```
# comments are permitted
AllQueueManagers:
User=abc
OPW=%^&aervrgtsr

QueueManager:
Name=QMA
User=user1
OPW=H&^dbgfh

Force=TRUE

QueueManager:
Name=QMB
User=user2
password=passw0rd
```

Notes:

1. The individual queue manager definitions take precedence over the global setting.
2. Attributes are case insensitive.

Constraints

When this exit is in use, the local user ID of the person running the application does not flow from the client to the server. The only identity information available is from the ini file contents.

Therefore, you must configure the queue manager to either use **ADOPTCTX(YES)**, or map the inbound connection request to an appropriate user ID through one of the available mechanisms, for example, "Channel authentication records" on page 51.

Important: If you add new passwords, or update old ones, the **runmqccred** command only processes any plain text passwords, leaving your obfuscated ones untouched.

Debugging

The exit writes to the standard IBM MQ trace when that is enabled.

To assist in debugging configuration issues, the exit can also write directly to stdout.

No channel security exit data (**SCYDATA**) configuration is normally required for the channel. However, you can specify:

ERROR
Only print information abut error conditions, such as not being able to find the configuration file.

DEBUG
Displays these error conditions, and some additional trace statements.

NOCHECKS
Bypasses the constraints on file permissions, and the further constraint that the `.ini` file should not contain any unprotected passwords.

You can put one or more of these elements into the **SCYDATA** field, separated by commas, in any order. For example, `SCYDATA=(NOCHECKS,DEBUG)`.

Note that the items are case-sensitive, and must be entered in uppercase.

Using `mqccred`

Once you have your file set up, you can invoke the channel exit by updating your client-connection channel definition to include the `SCYEXIT('mqccred(ChlExit)')` attribute:

```
DEFINE CHANNEL(channelname) CHLTYPE(clntconn) +
CONNAME(remote machine) +
QMNAME(remote qmgr) +
SCYEXIT('mqccred(ChlExit)') +
REPLACE
```

Related reference
SCYDATA
SCYEXIT
runmqccred

Connection authentication with the Java client

Connection authentication is a feature in IBM MQ that enables you to configure queue managers so that the queue manager can authenticate applications using a provided user ID and password. When the application is a Java application that is using client transport, connection authentication can be run in compatibility mode or MQCSP authentication mode.

The user ID and password to be authenticated is specified by the application using one of the following methods:

- In an IBM MQ classes for Java application, in the `MQEnvironment` class, or the properties Hashtable that is passed to the `com.ibm.mq.MQQueueManager` constructor.
- In an IBM MQ classes for JMS application, as arguments to the `createConnection(String username, String Password)` or `createContext(String username, String password)` method.

MQCSP authentication mode

In this mode, the client-side user ID that the application runs under is sent to the queue manager, as well as the user ID and password to be authenticated. The IBM MQ classes for Java and IBM MQ classes for JMS send the user ID and password to be authenticated to the queue manager in an MQCSP structure.

The user ID and password are available to a server-connection security exit within the MQCSP structure. The MQCSP structure address can be found in the **`SecurityParms`** field of the MQCXP structure for the channel.

MQCSP authentication mode has the following benefits:

- The maximum length of the user ID to be authenticated is 1024 characters.
- The maximum length of the password for authentication is 256 characters.
- Authorization checks for access to use IBM MQ resources can be performed using the client-side user ID that the application runs under, when the authentication information object that is used to control connection authentication on the queue manager is configured with ADOPTCTX(NO).

Compatibility mode

Before IBM MQ 8.0, the Java client could send a user ID and password across the client-connection channel to the server-connection channel, and have them provided to a security exit in the **`RemoteUserIdentifier`** and **`RemotePassword`** fields of the MQCD structure. In compatibility mode, this behavior is retained.

You might use this mode in combination with connection authentication, and migrate away from any security exits that were previously used to do the same job.

This mode has the following restrictions:

- The length of the user ID and password must be 12 characters or less. User IDs longer than 12 characters are truncated to 12 characters. This might cause the connection to fail with reason code MQRC_NOT_AUTHORIZED.
- The client-side user ID that the application runs under is not sent to the queue manager. You must either set ADOPTCTX(YES) on the authentication information object that is used to control connection authentication on the queue manager, or use another method, such as a channel authentication rule based on a TLS certificate, to set the channel MCA user ID that is checked for authorization to use IBM MQ resources.

Default authentication mode

The default authentication mode that is used by an IBM MQ classes for Java or IBM MQ classes for JMS client application varies depending on whether the application specifies a user ID and a password.

- If a user ID and password is specified, MQCSP authentication is used by default.
- If a user ID, but no password is specified, compatibility mode is used by default.
- If no user ID is specified, compatibility mode is always used.

In cases where a user ID is specified, a specific authentication mode can be chosen by the application for each individual connection, or set globally before the application is started, as described in “Choosing the authentication mode” on page 84.

Note: Applications that use the IBM MQ classes for JMS might be affected by the change to the default authentication mode in IBM MQ 9.3.0. After upgrading the IBM MQ classes for JMS to IBM MQ 9.3.0, applications that previously used compatibility mode by default will use MQCSP authentication instead. This might cause applications that previously connected successfully to a queue manager to fail to connect with a `JMSException` containing reason code 2035 (MQRC_NOT_AUTHORIZED). If this occurs, use one of the methods described in “Choosing the authentication mode” on page 84 to specify that the application uses compatibility mode.

Java applications that connect to the queue manager using local bindings always use MQCSP authentication mode.

Choosing the authentication mode

The authentication mode that is used by Java client applications that specify a user ID when connecting to the queue manager can be specified by using one of the following methods. These methods are listed in decreasing order of precedence. If the authentication mode is not specified using any of these methods, then the default authentication mode is used.

Note: The use of these methods to select the authentication mode was clarified in IBM MQ 9.3.0. In some cases, the authentication mode used by a Java client application might change when the IBM MQ classes for Java or IBM MQ classes for JMS are upgraded to IBM MQ 9.3.0. This might cause applications that previously connected successfully to a queue manager to fail to connect with a `JMSException` containing reason code 2035 (MQRC_NOT_AUTHORIZED). If this occurs, use one of the following methods to select the authentication mode that is required.

- Specify the authentication mode for each individual connection by setting the appropriate property in the application before connecting to the queue manager.
  - When using IBM MQ classes for Java, set the property *MQConstants.USE_MQCSP_AUTHENTICATION_PROPERTY* in the properties Hashtable that is passed to the `com.ibm.mq.MQQueueManager` constructor.
  - When using IBM MQ classes for JMS, set the property *JmsConstants.USER_AUTHENTICATION_MQCSP* on the appropriate connection factory before creating the connection.

  Set the value of these properties to one of the following values:

  true
  Use MQCSP authentication mode when authenticating with a queue manager.

  false
  Use compatibility mode when authenticating with a queue manager.

- Specify the authentication mode for all client connections made by an application by setting the *com.ibm.mq.cfg.jmqi.useMQCSPauthentication* Java system property when starting the application. Set the value of the property to one of the following values:

  Y
  Use MQCSP authentication mode when authenticating with a queue manager.

  N
  Use compatibility mode when authenticating with a queue manager.

  For example, the following command sets the property to select compatibility mode and starts a Java application:

  ```
  java -Dcom.ibm.mq.cfg.jmqi.useMQCSPauthentication=N application_name
  ```

- Specify the authentication mode for all client connections made by applications started in the same environment by setting the *com.ibm.mq.jmqi.useMQCSPauthentication* environment variable in the environment where the application is started. Set the value of the environment variable to one of the following values:

  Y
  Use MQCSP authentication mode when authenticating with a queue manager.

  N
  Use compatibility mode when authenticating with a queue manager.

- Specify the authentication mode for all applications that use a specific IBM MQ MQI client client configuration file by specifying the **`useMQCSPauthentication`** attribute in the JMQI stanza of the client configuration file. Set the value of the attribute to one of the following values:

YES
: Use MQCSP authentication mode when authenticating with a queue manager.

NO
: Use compatibility mode when authenticating with a queue manager.

For more information about the **`useMQCSPauthentication`** attribute, see JMQI stanza of the client configuration file.

Choosing authentication mode in IBM MQ Explorer

The IBM MQ Explorer is a Java application, so these two modes, compatibility mode and MQCSP authentication mode, are applicable to it as well.

MQCSP authentication mode is the default.

On panels where user identification is provided, there is a check box to enable or disable compatibility mode:

- By default, this check box is not selected. To use compatibility mode, select this check box.

Related concepts

"Connection authentication" on page 70
Connection authentication allows applications to supply authentication credentials when they connect to a queue manager. The queue manager validates the credentials. The user ID supplied in the credentials can also be adopted for use in authorization checks for resources that the application accesses.

"Connection authentication: Application changes" on page 76

"Connection authentication: User repositories" on page 78
For each of your queue managers, you can choose different types of authentication information object for authenticating user IDs and passwords.

Message security in IBM MQ

Message security in IBM MQ infrastructure is provided by Advanced Message Security.

Advanced Message Security (AMS) expands IBM MQ security services to provide data signing and encryption at the message level. The expanded services guarantees that message data has not been modified between when it is originally placed on a queue and when it is retrieved. In addition, AMS verifies that a sender of message data is authorized to place signed messages on a target queue.

Related tasks

"Advanced Message Security" on page 583
Advanced Message Security (AMS) is a component of IBM MQ that provides a high level of protection for sensitive data flowing through the IBM MQ network, while not impacting the end applications.

Planning for your security requirements

This collection of topics explains what you need to consider when planning security in an IBM MQ environment.

You can use IBM MQ for a wide variety of applications on a range of platforms. The security requirements are likely to be different for each application. For some, security will be a critical consideration.

IBM MQ provides a range of link-level security services, including support for Transport Layer Security (TLS).

You must consider certain aspects of security when planning to install IBM MQ:

- Multi On Multiplatforms, if you ignore these aspects and do nothing, you cannot use IBM MQ.
- z/OS On z/OS, the effect of ignoring these aspects is that your IBM MQ resources are unprotected. That is, all users can access and change all IBM MQ resources.

Authority to administer IBM MQ

IBM MQ administrators need authority to:

- Issue commands to administer IBM MQ
- Use the IBM MQ Explorer
- IBM i Use IBM i administrative panels and commands.
- z/OS Use the operations and control panels on z/OS
- z/OS Use the IBM MQ utility program, CSQUTIL, on z/OS
- z/OS Access the queue manager data sets on z/OS

For more information, see:

- ALW "Authority to administer IBM MQ on AIX, Linux, and Windows" on page 392
- IBM i "Authority to administer IBM MQ on IBM i" on page 90
- z/OS "Authority to administer IBM MQ on z/OS" on page 91

Authority to work with IBM MQ objects

Applications can access the following IBM MQ objects by issuing MQI calls:

- Queue managers
- Queues
- Processes
- Namelists
- Topics

Applications can also use Programmable Command Format (PCF) commands to access these IBM MQ objects, and to access channels and authentication information objects as well. These objects can be protected by IBM MQ so that the user IDs associated with the applications need authority to access them.

For more information, see "Authorization for applications to use IBM MQ" on page 93.

Channel security

The user IDs associated with message channel agents (MCAs) need authority to access various IBM MQ resources. For example, an MCA must be able to connect to a queue manager. If it is a sending MCA, it must be able to open the transmission queue for the channel. If it is a receiving MCA, it must be able to open destination queues. The user IDs associated with applications which need to administer channels, channel initiators, and listeners need authority to use the relevant PCF commands. However, most applications do not need such access.

For more information, see "Channel authorization" on page 113.

Additional considerations

You need to consider the following aspects of security only if you are using certain IBM MQ function or base product extensions:

- "Security for queue manager clusters" on page 125
- "Security for IBM MQ Publish/Subscribe" on page 126

Planning identification and authentication

Decide what user IDs to use, and how and at what levels you want to apply authentication controls.

You must decide how you will identify the users of your IBM MQ applications, bearing in mind that different operating systems support user IDs of different lengths. You can use channel authentication records to map from one user ID to another, or to specify a user ID based on some attribute of the connection. IBM MQ channels using TLS use digital certificates as a mechanism for identification and authentication. Each digital certificate has a subject distinguished name which can be mapped onto specific identities using channel authentication records. Additionally, CA certificates in the key repository determine which digital certificates may be used to authenticate to IBM MQ. For more information see:

- "Mapping a remote queue manager to an MCAUSER user ID" on page 378
- "Mapping a client user ID to an MCAUSER user ID" on page 378
- "Mapping an SSL or TLS Distinguished Name to an MCAUSER user ID" on page 379
- "Mapping an IP address to an MCAUSER user ID" on page 381

Planning authentication for a client application

You can apply authentication controls at four levels: at the communications level, in security exits, with channel authentication records, and in terms of the identification that is passed to a security exit.

There are four levels of security to consider. The diagram shows an IBM MQ MQI client that is connected to a server. Security is applied at four levels, as described in the following text. MCA is a Message Channel Agent.

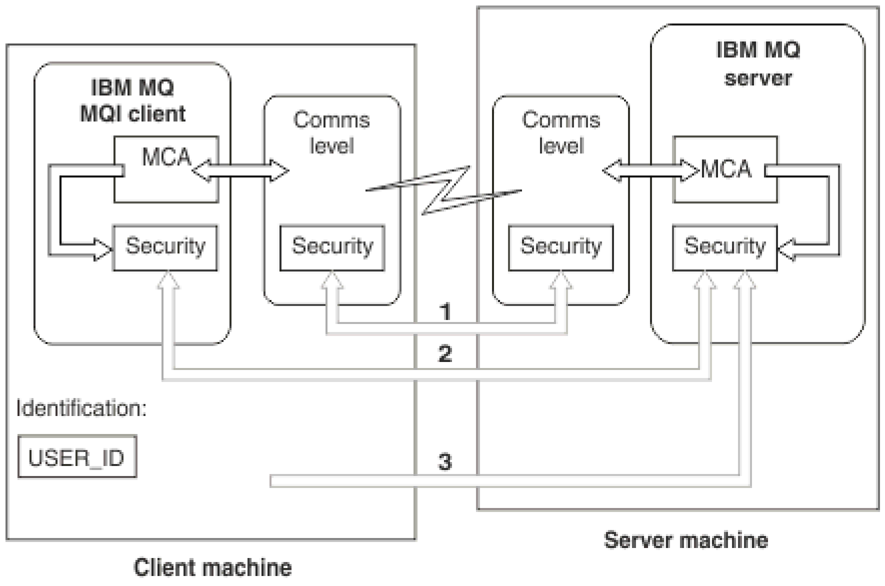


Figure 9. Security in a client/server connection

1. Communications level

   See arrow 1. To implement security at the communications level, use TLS. For more information, see "Cryptographic security protocols: TLS" on page 18

2. Channel authentication records

   See arrows 2 & 3. Authentication can be controlled by using the IP address or TLS distinguished names at the security level. A user ID can also be blocked or an asserted user ID can be mapped to a valid user ID. A full description is given in "Channel authentication records" on page 51.

3. Connection authentication

   See arrow 3. The client sends a user ID and a password, or an authentication token. For more information, see "Connection authentication: Configuration" on page 72.

4. Channel security exits

   See arrow 2. The channel security exits for client to server communication can work in the same way as for server to server communication. A protocol independent pair of exits can be written to provide mutual authentication of both the client and the server. A full description is given in Channel security exit programs.

5. Identification that is passed to a channel security exit

   See arrow 3. In client to server communication, the channel security exits do not have to operate as a pair. The exit on the IBM MQ client side can be omitted. In this case, the user ID is placed in the channel descriptor (MQCD) and the server-side security exit can alter it, if required.

   IBM MQ MQI clients also send extra information to assist identification.

   - The user ID that is passed to the server is the currently logged-on user ID on the client.
   - The security ID of the currently logged-on user.

   The values of the user ID and, if available, the security ID, can be used by the server security exit to establish the identity of the IBM MQ MQI client.

From IBM MQ 8.0, you can send passwords that are included in the MQCSP structure.

Linux V 9.4.0 AIX From IBM MQ 9.3.4, IBM MQ MQI clients connecting to IBM MQ queue managers running on AIX or Linux systems can also send authentication tokens in the MQCSP structure.

Warning: In some cases, the password or authentication token in an MQCSP structure for a client application is sent across the network in plain text. To ensure that client application passwords and authentication tokens are protected appropriately, see "MQCSP password protection" on page 31.

User IDs

When you create user IDs for client applications, the user IDs must not be longer than the maximum permitted length. You must not use the reserved user IDs UNKNOWN and NOBODY. If the server that the client connects to is an IBM MQ for Windows server, you must escape the use of the at sign, @. The permitted length of user IDs is dependent on the platform that is used for the server:

- z/OS Linux AIX On z/OS, AIX and Linux, the maximum length of a user ID is 12 characters.
- IBM i On IBM i, the maximum length of a user ID is 10 characters.
- Windows On Windows, if both the IBM MQ MQI client and the IBM MQ server are on Windows, and the server has access to the domain on which the client user ID is defined, the maximum length of a user ID is 20 characters. However, if the IBM MQ server is not a Windows server, the user ID is truncated to 12 characters.
- If you use the MQCSP structure to pass credentials, the maximum length of a user ID is 1024 characters. The MQCSP structure user ID cannot be used to circumvent the maximum userid length used by IBM MQ for authorization. For more information about the MQCSP structure, see "Identifying and authenticating users using the MQCSP structure" on page 314.

On AIX and Linux systems the default is that user IDs are used to authenticate, and groups are used for authorization. However, you can configure these systems to authorize against user Ids. For more information, see "OAM user-based permissions on AIX and Linux" on page 345. Windows systems can use both user IDs for both authentication and authorization and groups for authorization.

If you create service accounts, without paying attention to groups, and authorize all the user IDs differently, every user can access the information of every other user.

Restricted user IDs

The user IDs `UNKNOWN` and group `NOBODY` have special meanings to IBM MQ. Creating a user ID in the operating system called `UNKNOWN` or a group called `NOBODY` could have unintended results.

User IDs when connecting to an IBM MQ for Windows server

Windows

An IBM MQ for Windows server does not support the connection of an IBM MQ MQI client if the client is running under a user ID that contains the @ character, for example, `abc@d`. The return code to the `MQCONN` call at the client is MQRC_NOT_AUTHORIZED.

However, you can specify the user ID using two @ characters, for example, `abc@@d`. Using the `id@domain` format is the preferred practice, to ensure that the user ID is resolved in the correct domain consistently; thus `abc@@d@domain`.

Planning authorization

Plan the users who will have administrative authority and plan how to authorize users of applications to appropriately use IBM MQ objects, including those connecting from an IBM MQ MQI client.

Individuals or applications must be granted access in order to use IBM MQ. What access they require depend on the roles they undertake and the tasks which they need to perform. Authorization in IBM MQ can be subdivided into two main categories:

- Authorization to perform administrative operations
- Authorization for applications to use IBM MQ

Both classes of operation are controlled by the same component and an individual can be granted authority to perform both categories of operation.

The following topics give further information about specific areas of authorization that you must consider:

Authority to administer IBM MQ

IBM MQ administrators need authority to perform various functions. This authority is obtained in different ways on different platforms.

IBM MQ administrators need authority to:

- Issue commands to administer IBM MQ.
- Windows Linux Use the IBM MQ Explorer.
- z/OS Use the operations and control panels on z/OS.
- z/OS Use the IBM MQ utility program, CSQUTIL, on z/OS.
- z/OS Access the queue manager data sets on z/OS.

For more information, see the topic appropriate to your operating system.

ALW *Authority to administer IBM MQ on AIX, Linux, and Windows systems*

An IBM MQ administrator is a member of the mqm group. This group has access to all IBM MQ resources and can issue IBM MQ control commands. An administrator can grant specific authorities to other users.

To be an IBM MQ administrator on AIX, Linux, and Windows systems, a user must be a member of the *mqm group*. This group is created automatically when you install IBM MQ. To allow users to issue control commands, you must add them to the mqm group. This includes the root user on AIX and Linux.

Users who are not member of the mqm group can be granted administrative privileges, but they are not able to issue IBM MQ control commands, and they are authorized to execute only the commands for which they have been granted access.

Additionally, on Windows systems, the SYSTEM and Administrator accounts have full access to IBM MQ resources.

All members of the mqm group have access to all IBM MQ resources on the system, including being able to administer any queue manager running on the system. This access can be revoked only by removing a user from the mqm group. On Windows systems, members of the Administrators group also have access to all IBM MQ resources.

Administrators can use the control command **runmqsc** to issue IBM MQ Script (MQSC) commands. When **runmqsc** is used in indirect mode to send MQSC commands to a remote queue manager, each MQSC command is encapsulated within an Escape PCF command. Administrators must have the required authorities for the MQSC commands to be processed by the remote queue manager.

The IBM MQ Explorer issues PCF commands to perform administration tasks. Administrators require no additional authorities to use the IBM MQ Explorer to administer a queue manager on the local system. When the IBM MQ Explorer is used to administer a queue manager on another system, administrators must have the required authorities for the PCF commands to be processed by the remote queue manager.

For more information about the authority checks carried out when PCF and MQSC commands are processed, see the following topics:

- For commands that operate on queue managers, queues, channels, processes, namelists, and authentication information objects, see "Authorization for applications to use IBM MQ" on page 93.
- For commands that operate on channels, channel initiators, listeners, and clusters, see Channel security.
- z/OS For MQSC commands that are processed by the command server on IBM MQ for z/OS, see "Command security and command resource security on z/OS" on page 91.

For more information about the authority you need to administer IBM MQ for AIX, Linux, and Windows systems, see the related information.

IBM i *Authority to administer IBM MQ on IBM i*

To be an IBM MQ administrator on IBM i, you must be a member of the *QMQMADM group*. This group has properties similar to those of the mqm group on AIX, Linux, and Windows systems. In particular, the QMQMADM group is created when you install IBM MQ for IBM i, and members of the QMQMADM group have access to all IBM MQ resources on the system. You also have access to all IBM MQ resources if you have *ALLOBJ authority.

Administrators can use CL commands to administer IBM MQ. One of these commands is GRTMQMAUT, which is used to grant authorities to other users. Another command, STRMQMMQSC, enables an administrator to issue MQSC commands to a local queue manager.

There are two groups of CL command provided by IBM MQ for IBM i:

Group 1
To issue a command in this category, a user must be a member of the QMQMADM group or have *ALLOBJ authority. GRTMQMAUT and STRMQMMQSC belong to this category, for example.

Group 2
To issue a command in this category, a user does not need to be a member of the QMQMADM group or have *ALLOBJ authority. Instead, two levels of authority are required:

- The user requires IBM i authority to use the command. This authority is granted by using the GRTOBJAUT command.
- The user requires IBM MQ authority to access any IBM MQ object associated with the command. This authority is granted by using the GRTMQMAUT command.

The following examples show commands in this group:

- CRTMQMQ, Create MQM Queue
- CHGMQMPRC, Change MQM Process
- DLTMQMNL, Delete MQM Namelist
- DSPMQMAUTI, Display MQM Authentication Information
- CRTMQMCHL, Create MQM channel

For more information about this group of commands, see "Authorization for applications to use IBM MQ" on page 93.

For a complete list of group 1 and group 2 commands, see "Access authorities for IBM MQ objects on IBM i" on page 157

For more information about the authority you need to administer IBM MQ on IBM i, see Administering IBM i .

z/OS Authority to administer IBM MQ on z/OS

This collection of topics describes various aspects of the authority you need to administer IBM MQ for z/OS.

z/OS *Authority checks on z/OS*

IBM MQ for z/OS uses the System Authorization Facility (SAF) to route requests for authority checks to an external security manager (ESM) such as the z/OS Security Server Resource Access Control Facility (RACF). IBM MQ does no authority checks of its own.

It is assumed that you are using RACF as your ESM. If you are using a different ESM, you might need to interpret the information provided for RACF in a way that is relevant to your ESM.

You can specify whether you want authority checks turned on or off for each queue manager individually or for every queue manager in a queue sharing group. This level of control is called *subsystem security*. If you turn subsystem security off for a particular queue manager, no authority checks are carried out for that queue manager.

If you turn subsystem security on for a particular queue manager, authority checks can be performed at two levels:

Queue sharing group level security
Authority checks use RACF profiles that are shared by all queue managers in the queue sharing group. This means that there are fewer profiles to define and maintain, making security administration easier.

Queue manager level security
Authority checks use RACF profiles specific to the queue manager.

You can use a combination of queue sharing group and queue manager level security. For example, you can arrange for profiles specific to a queue manager to override those of the queue sharing group to which it belongs.

Subsystem security, queue sharing group level security, and queue manager level security are turned on or off by defining *switch profiles*. A switch profile is a normal RACF profile that has a special meaning to IBM MQ.

z/OS *Command security and command resource security on z/OS*

Command security relates to the authority to issue a command; command resource authority relates to the authority to perform an operation on a resource. Both are implemented y using RACF classes.

Authority checks are carried out when an IBM MQ administrator issues an MQSC command. This is called *command security*.

To implement command security, you must define certain RACF profiles and give the necessary groups and user IDs access to these profiles at the required levels. The name of a profile for command security contains the name of an MQSC command.

Some MQSC commands perform an operation on an IBM MQ resource, such as the DEFINE QLOCAL command to create a local queue. When an administrator issues an MQSC command, authority checks are carried out to determine whether the requested operation can be performed on the resource specified in the command. This is called *command resource security*.

To implement command resource security, you must define certain RACF profiles and give the necessary groups and user IDs access to these profiles at the required levels. The name of a profile for command resource security contains the name of an IBM MQ resource and its type (QUEUE, PROCESS, NAMELIST, TOPIC, AUTHINFO, or CHANNEL).

Command security and command resource security are independent. For example, when an administrator issues the command:

```
DEFINE QLOCAL(MOON.EUROPA)
```

the following authority checks are performed:

- Command security checks that the administrator is authorized to issue the DEFINE QLOCAL command.
- Command resource security checks that the administrator is authorized to perform an operation on the local queue called MOON.EUROPA.

Command security and command resource security can be turned on or off by defining switch profiles.

z/OS *MQSC commands and the system command input queue on z/OS*
Use this topic to understand how the command server processes MQSC commands directed to the system command input queue on z/OS.

Command security and command resource security are also used when the command server retrieves a message containing an MQSC command from the system command input queue. The user ID that is used for the authority checks is the one found in the *UserIdentifier* field in the message descriptor of the message containing the MQSC command. This user ID must have the required authorities on the queue manager where the command is processed. For more information about the *UserIdentifier* field and how it is set, see Message context.

Messages containing MQSC commands are sent to the system command input queue in the following circumstances:

- The operations and control panels send MQSC commands to the system command input queue of the target queue manager. The MQSC commands correspond to the actions you choose on the panels. The *UserIdentifier* field in each message is set to the TSO user ID of the administrator.
- The COMMAND function of the IBM MQ utility program, CSQUTIL, sends the MQSC commands in the input data set to the system command input queue of the target queue manager. The COPY and EMPTY functions send DISPLAY QUEUE and DISPLAY STGCLASS commands. The *UserIdentifier* field in each message is set to the job user ID.
- The MQSC commands in the CSQINPX data sets are sent to the system command input queue of the queue manager to which the channel initiator is connected. The *UserIdentifier* field in each message is set to the channel initiator address space user ID.

  No authority checks are performed when MQSC commands are issued from the CSQINP1 and CSQINP2 data sets. You can control who is allowed to update these data sets using RACF data set protection.
- Within a queue sharing group, a channel initiator might send START CHANNEL commands to the system command input queue of the queue manager to which it is connected. A command is sent when an outbound channel that uses a shared transmission queue is started by triggering. The *UserIdentifier* field in each message is set to the channel initiator address space user ID.
- An application can send MQSC commands to a system command input queue. By default, the *UserIdentifier* field in each message is set to the user ID associated with the application.
- On AIX, Linux, and Windows systems, the **runmqsc** control command can be used in indirect mode to send MQSC commands to the system command input queue of a queue manager on z/OS. The *UserIdentifier* field in each message is set to the user ID of the administrator who issued the **runmqsc** command.

z/OS *Access to the queue manager data sets on z/OS*
IBM MQ for z/OS administrators need authority to access the queue manager data sets. Use this topic to understand which data sets need RACF protection.

These data sets include:

- The data sets referred to by CSQINP1, CSQINP2, and CSQINPT in the started task procedure of the queue manager.
- The queue manager's page sets, active log data sets, archive log data sets, and bootstrap data sets (BSDSs)
- The data sets referred to by CSQXLIB and CSQINPX in the channel initiator's started task procedure

You must protect the data sets so that no unauthorized user can start a queue manager or gain access to any queue manager data. To do this, use RACF data set protection.

Authorization for applications to use IBM MQ

When applications access objects, the user IDs associated with the applications need appropriate authority.

Applications can access the following IBM MQ objects by issuing MQI calls:

- Queue managers
- Queues
- Processes
- Namelists
- Topics

Applications can also use PCF commands to administer IBM MQ objects. When the PCF command is processed, it uses the authority context of the user ID that put the PCF message.

Applications, in this context, include those written by users and vendors, and those supplied with IBM MQ for z/OS.

z/OS The applications supplied with IBM MQ for z/OS include:

- The operations and control panels
- The IBM MQ utility program, CSQUTIL
- The dead letter queue handler utility, CSQUDLQH

Applications that use IBM MQ classes for Java, IBM MQ classes for JMS, IBM MQ classes for .NET, or the Message Service Clients for C/C++ and .NET use the MQI indirectly.

MCAs also issue MQI calls and the user IDs associated with the MCAs need authority to access these IBM MQ objects. For more information about these user IDs and the authorities they require, see "Channel authorization" on page 113.

z/OS On z/OS, applications can also use MQSC commands to access these IBM MQ objects but command security and command resource security provide the authority checks in these circumstances. z/OS For more information, see "Command security and command resource security on z/OS" on page 91 and "MQSC commands and the system command input queue on z/OS" on page 92.

IBM i On IBM i, a user that issues a CL command in Group 2 might require authority to access an IBM MQ object associated with the command. For more information, see "When authority checks are performed" on page 93.

When authority checks are performed

Authority checks are performed when an application attempts to access a queue manager, queue, process, or namelist.

On IBM i, authority checks might also be performed when a user issues a CL command in Group 2 that accesses any of these IBM MQ objects. The checks are performed in the following circumstances:

When an application connects to a queue manager using an MQCONN or MQCONNX call

The queue manager asks the operating system for the user ID associated with the application. The queue manager then checks that the user ID is authorized to connect to it and retains the user ID for future checks.

Users do not have to sign on to IBM MQ. IBM MQ assumes that users are signed on to the underlying operating system and are been authenticated by it.

When an application opens an IBM MQ object using an MQOPEN or MQPUT1 call

All authority checks are performed when an object is opened, not when it is accessed later. For example, authority checks are performed when an application opens a queue. They are not performed when the application puts messages on the queue or gets messages from the queue.

When an application opens an object, it specifies the types of operation it needs to perform on the object. For example, an application might open a queue to browse the messages on it, get messages from it, but not to put messages on it. For each type of operation, the queue manager checks that the user ID associated with the application has the authority to perform that operation.

When an application opens a queue, the authority checks are performed against the object named in the `ObjectName` field of the object descriptor. The `ObjectName` field is used on the `MQOPEN` or `MQPUT1` calls. If the object is an alias queue or a remote queue definition, the authority checks are performed against the object itself. They are not performed on the queue to which the alias queue or the remote queue definition resolves. This means that the user does not need permission to access it. Limit the authority to create queues to privileged users. If you do not, users might bypass the normal access control simply by creating an alias.

An application can reference a remote queue explicitly. It sets the `ObjectName` and `ObjectQMgrName` fields in the object descriptor to the names of the remote queue and the remote queue manager. The authority checks are performed against the transmission queue with the same name as the remote queue manager:

- z/OS On z/OS, a check is made on the RACF queue profile that matches the remote queue manager name, and is performed whether or not this transmission queue is locally defined.
- Multi On Multiplatforms, a check is made against the RQMNAME profile that matches the remote queue manager name, if clustering is being used.

An application can reference a cluster queue explicitly by setting the `ObjectName` field in the object descriptor to the name of the cluster queue. The authority checks are performed against the cluster transmission queue, `SYSTEM.CLUSTER.TRANSMIT.QUEUE`.

The authority to a dynamic queue is based on the model queue from which it is derived, but is not necessarily the same; see note 1.

The user ID that the queue manager uses for the authority checks is obtained from the operating system. The user ID is obtained when the application connects to the queue manager. A suitably authorized application can issue an `MQOPEN` call specifying an alternative user ID; access control checks are then made on the alternative user ID. Using an alternate user ID does not change the user ID associated with the application, only the one used for access control checks.

When an application subscribes to a topic using an MQSUB call

When an application subscribes to a topic, it specifies the type of operation that it needs to perform. It is either creating a subscription, altering an existing subscription, or resuming an existing subscription without changing it. For each type of operation, the queue manager checks that the user ID that is associated with the application has the authority to perform the operation.

When an application subscribes to a topic, the authority checks are performed against topic objects that are found in the topic tree. The topic objects are at, or above, the point in the topic tree at which the application subscribed. The authority checks might involve checks on more than one topic object. The user ID that the queue manager uses for the authority checks is obtained from the operating system. The user ID is obtained when the application connects to the queue manager.

The queue manager performs authority checks on subscriber queues but not on managed queues.

When an application deletes a permanent dynamic queue using an MQCLOSE call
The object handle specified on the MQCLOSE call is not necessarily the same one returned by the MQOPEN call that created the permanent dynamic queue. If it is different, the queue manager checks the user ID associated with the application that issued the MQCLOSE call. It checks that the user ID is authorized to delete the queue.

When an application that closes a subscription to remove it did not create it, the appropriate authority is required to remove it.

When a PCF command that operates on an IBM MQ object is processed by the command server
This rule includes the case where a PCF command operates on an authentication information object.

The user ID that is used for the authority checks is the one found in the `UserIdentifier` field in the message descriptor of the PCF command. This user ID must have the required authorities on the queue manager where the command is processed. The equivalent MQSC command encapsulated within an Escape PCF command is treated in the same way. For more information about the `UserIdentifier` field, and how it is set, see "Message context" on page 95.

IBM i **On IBM i, when a user issues a CL command in Group 2 that operates on an IBM MQ object**
This rule includes the case where a CL command in Group 2 operates on an authentication information object.

Checks are performed to determine whether the user has the authority to operate on an IBM MQ object associated with the command. The checks are performed unless the user is a member of the QMQMADM group or has *ALLOBJ authority. The authority required depends on the type of operation that the command performs on the object. For example, the command **CHGMQMQ**, Change MQM Queue, requires the authority to change the attributes of the queue specified by the command. In contrast, the command **DSPMQMQ**, Display MQM Queue, requires the authority to display the attributes of the queue specified by the command.

Many commands operate on more than one object. For example, to issue the command **DLTMQMQ**, Delete MQM Queue, the following authorities are required:

- The authority to connect to the queue manager specified by the command
- The authority to delete the queue specified by the command

Some commands operate on no object at all. In this case, the user requires only IBM i authority to issue one of these commands. **STRMQMLSR**, Start MQM Listener, is an example of such a command.

Alternate user authority

When an application opens an object or subscribes to a topic, the application can supply a user ID on the MQOPEN, MQPUT1, or MQSUB call. It can ask the queue manager to use this user ID for authority checks instead of the one associated with the application.

The application succeeds in opening the object only if both the following conditions are met:

- The user ID associated with the application has the authority to supply a different user ID for authority checks. The application is said to have *alternate user authority*.
- The user ID supplied by the application has the authority to open the object for the types of operation requested, or to subscribe to the topic.

Message context

Message context information allows the application that retrieves a message to find out about the originator of the message. The information is held in fields in the message descriptor and the fields are divided into three logical parts

These parts are as follows:

identity context
These fields contain information about the user of the application that put the message on the queue.

origin context
These fields contain information about the application itself and when the message was put on the queue.

user context
These fields contain message properties that applications can use to select messages that the queue manager should deliver.

When an application puts a message on a queue, the application can ask the queue manager to generate the context information in the message. This is the default action. Alternatively, it can specify that the context fields are to contain no information. The user ID associated with an application requires no special authority to do either of these.

An application can set the identity context fields in a message, allowing the queue manager to generate the origin context, or it can set all the context fields. An application can also pass the identity context fields from a message it has retrieved to a message it is putting on a queue, or it can pass all the context fields. However, the user ID associated with an application requires authority to set or pass context information. An application specifies that it intends to set or pass context information when it opens the queue on which it is about to put messages, and its authority is checked at this time.

Here is a brief description of each of the context fields:

Identity context

UserIdentifier
The user ID associated with the application that put the message. If the queue manager sets this field, it is set to the user ID obtained from the operating system when the application connects to the queue manager.

AccountingToken
Information that can be used to charge for the work done as a result of the message.

ApplIdentityData
If the user ID associated with an application has authority to set the identity context fields, or to set all the context fields, the application can set this field to any value related to identity. If the queue manager sets this field, it is set to blank.

Origin context

PutApplType
The type of the application that put the message; a CICS® transaction, for example.

PutApplName
The name of the application that put the message.

PutDate
The date when the message was put.

PutTime
The time when the message was put.

ApplOriginData
If the user ID associated with an application has authority to set all the context fields, the application can set this field to any value related to origin. If the queue manager sets this field, it is set to blank.

User context

The following values are supported for **`MQINQMP`** or **`MQSETMP`**:

MQPD_USER _CONTEXT

The property is associated with the user context.

No special authorization is required to be able to set a property associated with the user context using the MQSETMP call.

On a V7.0 or subsequent queue manager, a property associated with the user context is saved as described for MQOO_SAVE_ALL_CONTEXT. An MQPUT with MQOO_PASS_ALL_CONTEXT specified causes the property to be copied from the saved context into the new message.

MQPD_NO_CONTEXT

The property is not associated with a message context.

An unrecognized value is rejected with MQRC_PD_ERROR. The initial value of this field is **MQPD_NO_CONTEXT**.

For a detailed description of each of the context fields, see MQMD - Message descriptor. For more information about how to use message context, see Message context.

IBM i ALW *Authority to work with IBM MQ objects on* IBM i *IBM i , AIX, Linux, and Windows systems*

The authorization service component provided with IBM MQ is called the *object authority manager* (OAM). It provides access control via authentication and authorization checks.

Authentication.

The authentication check performed by the OAM provided with IBM MQ is basic, and is only performed in specific circumstances. It is not intended to meet the strict requirements expected in a highly secure environment.

The OAM performs its authentication check when an application connects to a queue manager, and the following conditions are true:

- If an MQCSP structure has been supplied by the connecting application, and
- The *AuthenticationType* attribute in the MQCSP structure is given the value MQCSP_AUTH_USER_ID_AND_PWD, and
- The CHCKLOCL or CHKCCLNT value on the configured AUTHINFO object is not 'NONE'

The authentication steps in the OAM validate the password using operating system services, which might have been configured to perform additional checks, such as ensuring the username has not had too many incorrect password test attempts.

It is possible for alternative authentication mechanisms to be used if you write a new authorization service component, or obtain one from a vendor.

Authorization.

The authorization checks are comprehensive, and are intended to meet most normal requirements.

Authorization checks are performed when an application issues an MQI call to access a queue manager, queue, process, topic, or namelist. They are also performed at other times, for example, when a command is being performed by the Command Server.

On IBM i IBM i , AIX, Linux, and Windows systems, the *authorization service* provides the access control when an application issues an MQI call to access an IBM MQ object that is a queue manager, queue, process, topic, or namelist. This includes checks for alternative user authority and the authority to set or pass context information.

Windows On Windows , the OAM gives members of the Administrators group the authority to access all IBM MQ objects, even when UAC is enabled. Additionally, on Windows systems, the SYSTEM account has full access to IBM MQ resources.

The authorization service also provides authority checks when a PCF command operates on one of these IBM MQ objects or an authentication information object. The equivalent MQSC command encapsulated within an Escape PCF command is treated in the same way.

IBM i On IBM i , unless the user is a member of the QMQMADM group or has *ALLOBJ authority, the authorization service also provides authority checks when a user issues a CL command in Group 2 that operates on any of these IBM MQ objects or an authentication information object.

The authorization service is an *installable service*, which means that it is implemented by one or more *installable service components*. Each component is invoked using a documented interface. This enables users and vendors to provide components to augment or replace those provided by the IBM MQ products.

The authorization service component provided with IBM MQ is called the object authority manager (OAM). The OAM is automatically enabled for each queue manager you create.

The OAM maintains an access control list (ACL) for each IBM MQ object it is controlling access to. On AIX and Linux systems, only group IDs can appear in an ACL. This means that all members of a group have the same authorities. On IBM i IBM i and on Windows systems, both user IDs and group IDs can appear in an ACL. This means that authorities can be granted to individual users and groups.

A 12 character limitation applies to both the group and the user ID. UNIX platforms generally restrict the length of a user ID to 12 characters. AIX and Linux have raised this limit but IBM MQ continues to observe a 12 character restriction on all UNIX platforms. If you use a user ID of greater than 12 characters, IBM MQ replaces it with the value "UNKNOWN". Do not define a user ID with a value of "UNKNOWN".

The OAM can authenticate a user and change appropriate identity context fields. You enable this by specifying a connection security parameters structure (MQCSP) on an MQCONNX call. The structure is passed to the OAM Authenticate User function (MQZ_AUTHENTICATE_USER), which sets appropriate identity context fields. If an MQCONNX connection from an IBM MQ client, the information in the MQCSP is flowed to the queue manager to which the client is connecting over the client-connection and server-connection channel. If security exits are defined on that channel, the MQCSP is passed into each security exit and can be altered by the exit. Security exits can also create the MQCSP. For more details of the use of security exits in this context, see Channel security exit programs.

Warning: In some cases, the password in an MQCSP structure for a client application will be sent across a network in plain text. To ensure that client application passwords are protected appropriately, see IBM MQCSP password protection.

On AIX, Linux, and Windows systems, the control command **setmqaut** grants and revokes authorities and is used to maintain the ACLs. For example, the command:

```
setmqaut -m JUPITER -t queue -n MOON.EUROPA -g VOYAGER +browse +get
```

allows the members of the group VOYAGER to browse messages on the queue MOON.EUROPA that is owned by the queue manager JUPITER. It allows the members to get messages from the queue as well. To revoke these authorities later, enter the following command:

```
setmqaut -m JUPITER -t queue -n MOON.EUROPA -g VOYAGER -browse -get
```

The command:

```
setmqaut -m JUPITER -t queue -n MOON.* -g VOYAGER +put
```

allows the members of the group VOYAGER to put messages on any queue with a name that commences with the characters `MOON.`. MOON.* is the name of a generic profile. A *generic profile* allows you to grant authorities for a set of objects using a single **setmqaut** command.

The control command **dspmqaut** is available to display the current authorities that a user or group has for a specified object. The control command **dmpmqaut** is also available to display the current authorities associated with generic profiles.

IBM i On IBM i, an administrator uses the CL command GRTMQMAUT to grant authorities and the CL command RVKMQMAUT to revoke authorities. Generic profiles can be used as well. For example, the CL command:

```
GRTMQMAUT MQMNAME(JUPITER) OBJTYPE(*Q) OBJ('MOON.*') USER(VOYAGER) AUT(*PUT)
```

provides the same function as the previous example of a **`setmqaut`** command; it allows the members of the group VOYAGER to put messages on any queue with a name that commences with the characters `MOON.`

IBM i The CL command DSPMQMAUT displays the current authorities that user or group has for a specified object. The CL commands WRKMQMAUT and WRKMQMAUTD are also available to work with the current authorities associated with objects and generic profiles.

If you do not want any authority checks, for example, in a test environment, you can disable the OAM.

Multi *Using PCF to access OAM commands*
On IBM i, AIX, Linux, and Windows systems, you can use PCF commands to access OAM administration commands.

The PCF commands and their equivalent OAM commands are as follows:

Table 8. PCF commands and their equivalent OAM commands

PCF command	OAM command
Inquire Authority Records	dmpmqaut
Inquire Entity Authority	dspmqaut
Set Authority Record	setmqaut
Delete Authority Record	setmqaut with -remove option

The **`setmqaut`** and **`dmpmqaut`** commands are restricted to members of the mqm group. The equivalent PCF commands can be executed by users in any group who have been granted dsp and chg authorities on the queue manager.

For more information about using these commands, see Introduction to Programmable Command Formats.

z/OS *Authority to work with IBM MQ objects on z/OS*

On z/OS, there are seven categories of authority check associated with calls to the MQI. You must define certain RACF profiles and give appropriate access to these profiles. Use the *RESLEVEL* profile to control how many users IDs are checked.

The seven categories of authority check associated with calls to the MQI:

Connection security
The authority checks that are performed when an application connects to a queue manager

Queue security
The authority checks that are performed when an application opens a queue or deletes a permanent dynamic queue

Process security
The authority checks that are performed when an application opens a process object

Namelist security
The authority checks that are performed when an application opens a namelist object

Alternate user security
The authority checks that are performed when an application requests alternate user authority when opening an object

Context security
The authority checks that are performed when an application opens a queue and specifies that it intends to set or pass the context information in the messages it puts on the queue

Topic security
The authority checks that are performed when an application opens a topic

Each category of authority check is implemented in the same way that command security and command resource security are implemented. You must define certain RACF profiles and give the necessary groups and user IDs access to these profiles at the required levels. For queue security, the level of access determines the types of operation the application can perform on a queue. For context security, the level of access determines whether the application can:

- Pass all the context fields
- Pass all the context fields and set the identity context fields
- Pass and set all the context fields

Each category of authority check can be turned on or off by defining switch profiles.

All the categories, except connection security, are known collectively as *API-resource security*.

By default, when an API-resource security check is performed as a result of an MQI call from an application using a batch connection, only one user ID is checked. When a check is performed as a result of an MQI call from a CICS or IMS application, or from the channel initiator, two user IDs are checked.

By defining a *RESLEVEL profile*, however, you can control whether zero, one, or two users IDs are checked. The number of user IDs that are checked is determined by the user ID associated with the type of connection when an application connects to the queue manager and the access level that user ID has to the RESLEVEL profile. The user ID associated with each type of connection is:

- The user ID of the connecting task for batch connections
- The CICS address space user ID for CICS connections
- The IMS region address space user ID for IMS connections
- The channel initiator address space user ID for channel initiator connections

For more information about the authority to work with IBM MQ objects on z/OS, see "Authority to administer IBM MQ on z/OS" on page 91.

Security for remote messaging

This section deals with remote messaging aspects of security.

You must provide users with authority to use the IBM MQ facilities. This is organized according to actions to be taken with respect to objects and definitions. For example:

- Queue managers can be started and stopped by authorized users
- Applications must connect to the queue manager and have authority to use queues
- Message channels must be created and controlled by authorized users
- Objects are kept in libraries and access to these libraries can be restricted

The message channel agent at a remote site must check that the message being delivered originated from a user with authority to do so at this remote site. In addition, as MCAs can be started remotely, it might be necessary to verify that the remote processes trying to start your MCAs are authorized to do so. There are four possible ways for you to deal with this:

1. Make appropriate use of the PutAuthority attribute of your RCVR, RQSTR, or CLUSRCVR channel definition to control which user is used for authorization checks at the time incoming messages are put to your queues. See the DEFINE CHANNEL command description in the MQSC Command Reference.
2. Implement channel authentication records to reject unwanted connection attempts, or to set an MCAUSER value based on the following: the remote IP address, the remote user ID, the TLS Subject Distinguished Name (DN) provided, or the remote queue manager name.
3. Implement *user exit* security checking to ensure that the corresponding message channel is authorized. The security of the installation hosting the corresponding channel ensures that all users are properly authorized, so that you do not need to check individual messages.
4. Implement *user exit* message processing to ensure that individual messages are vetted for authorization.

IBM i *Security of IBM MQ for IBM i objects*

This section deals with remote messaging aspects of security.

You must provide users with authority to make use of the IBM MQ for IBM i facilities. This authority is organized according to actions to be taken with respect to objects and definitions. For example:

- Queue managers can be started and stopped by authorized users
- Applications need to connect to the queue manager, and have authority to make use of queues
- Message channels need to be created and controlled by authorized users

The message channel agent at a remote site must check that the message being delivered has derived from a user with authority to isue the message at this remote site. In addition, as MCAs can be started remotely, it might be necessary to verify that the remote processes trying to start your MCAs are authorized to do so. There are four possible ways for you to deal with this:

- Decree in the channel definition that messages must contain acceptable *context* authority, otherwise they are discarded.
- Implement channel authentication records to reject unwanted connection attempts, or to set an MCAUSER value based on one of the following: the remote IP address, the remote user ID, the TLS Distinguished Name (DN) provided, or the remote queue manager name.
- Implement user exit security checking to ensure that the corresponding message channel is authorized. The security of the installation hosting the corresponding channel ensures that all users are properly authorized, so that you do not need to check individual messages.
- Implement user exit message processing to ensure that individual messages are vetted for authorization.

Here are some facts about the way IBM MQ for IBM i operates security:

- Users are identified and authenticated by IBM i.
- Queue manager services invoked by applications are run with the authority of the queue manager user profile, but in the user's process.
- Queue manager services invoked by user commands are run with the authority of the queue manager user profile.

Linux AIX *Security of objects on AIX and Linux*

Administration users must be part of the mqm group on your system (including root) if this ID is going to use IBM MQ administration commands.

You should always run amqcrsta as the "mqm" user ID.

User IDs on AIX and Linux

The queue manager converts all uppercase or mixed case user identifiers into lowercase. The queue manager then inserts the user identifiers into the context part of a message, or checks their authorization. Authorizations are therefore based only on lowercase identifiers.

Windows *Security of objects on Windows systems*

Administration users must be part of both the mqm group and the administrators group on Windows systems if this ID is going to use IBM MQ administration commands.

User IDs on Windows systems

On Windows systems, *if there is no message exit installed*, the queue manager converts any uppercase or mixed case user identifiers into lowercase. The queue manager then inserts the user identifiers into the context part of a message, or checks their authorization. Authorizations are therefore based only on lowercase identifiers.

User IDs across systems

Platforms other than AIX, Linux, and Windows systems use uppercase characters for user IDs in messages. To allow AIX, Linux, and Windows systems to use lowercase user IDs in messages, the message channel agent (MCA) must carry out the appropriate conversions of alphabetic characters.

To allow AIX, Linux, and Windows systems to use lowercase user IDs in messages, the following conversions are carried out by the message channel agent (MCA) on these platforms:

At the sending end
The alphabetic characters in all user IDs are converted to uppercase characters, if there is no message exit installed.

At the receiving end
The alphabetic characters in all user IDs are converted to lowercase characters, if there is no message exit installed.

The automatic conversions are not carried out if you provide a message exit on AIX, Linux, and Windows for any other reason.

Using a custom authorization service

IBM MQ supplies an installable authorization service. You can choose to install an alternative service.

The authorization service component supplied with IBM MQ is called the Object Authority Manager (OAM). If the OAM does not supply the authorization facilities you need, you can write your own authorization service component. The installable service functions that must be implemented by an authorization service component are described at Installable services interface reference information.

Access control for clients

Access control is based on user IDs. There can be many user IDs to administer, and user IDs can be in different formats. You can set the server-connection channel property MCAUSER to a special user ID value for use by clients.

Access control in IBM MQ is based on user IDs. The user ID of the process making MQI calls is normally used. For MQ MQI clients, the server-connection MCA makes MQI calls on behalf of MQ MQI clients. You can select an alternative user ID for the server-connection MCA to use for making MQI calls. The alternative user ID can be associated either with the client workstation, or with anything you choose to organize and control the access of clients. The user ID needs to have the necessary authorities allocated to it on the server to issue MQI calls. Choosing an alternative user ID is preferable to allowing clients to make MQI calls with the authority of the server-connection MCA.

Table 9. The user ID used by a server-connection channel

User ID	When used
The user ID that is set by a security exit	Used unless blocked by a **CHLAUTH TYPE(BLOCKUSER)** rule. See the following section, “Setting the user ID in a security exit” on page 103 for more information.
The user ID that is set by a CHLAUTH rule	Used unless over-ridden by a security exit. See Channel Authentication Records for more information.
The user ID that is defined in the **MCAUSER** attribute in the SVRCONN channel definition	Used unless over-ridden by a security exit or a CHLAUTH rule.
The user ID that is flowed from the client machine	Used when no user ID is set by any other means.
The user ID that started the server-connection channel	Used when no user ID is set by any other means and no client user ID is flowed. See the following section, “The user ID that runs the channel program” on page 103 for more information.

Because the server-connection MCA makes MQI calls on behalf of remote users, it is important to consider the security implications of the server-connection MCA issuing MQI calls on behalf of remote clients and how to administer the access of a potentially large number of users.

- One approach is for the server-connection MCA to issue MQI calls on its own authority. But beware, it is normally undesirable for the server-connection MCA, with its powerful access capabilities, to issue MQI calls on behalf of client users.
- Another approach is to use the user ID that flows from the client. The server-connection MCA can issue MQI calls using the access capabilities of the client user ID. This approach presents a number of questions to consider:
  1. There are different formats for the user ID on different platforms. This sometimes causes problems if the format of the user ID on the client differs from the acceptable formats on the server.
  2. There are potentially many clients, with different, and changing user IDs. The IDs need to be defined and managed on the server.
  3. Is the user ID to be trusted? Any user ID can be flowed from a client, not necessarily the ID of the logged on user. For example, the client might flow an ID with full `mqm` authority that was intentionally only defined on the server for security reasons.
- The preferred approach is to define client identification tokens at the server, and so limit the capabilities of client connected applications. This is typically done by setting the server-connection channel property MCAUSER to a special user ID value to be used by clients, and defining few IDs for use by clients with different level of authorization on the server.

Setting the user ID in a security exit

For IBM MQ MQI clients, the process that issues the MQI calls is the server-connection MCA. The user ID used by the server-connection MCA is contained in either the `MCAUserIdentifier` or `LongMCAUserIdentifier` fields of the MQCD. The contents of these fields are set by:

- Any values set by security exits
- The user ID from the client
- MCAUSER (in the server-connection channel definition)

The security exit can override the values that are visible to it, when it is invoked.

- If the server-connection channel MCAUSER attribute is set to nonblank, the MCAUSER value is used.
- If the server-connection channel MCAUSER attribute is blank, the user ID received from the client is used.
- If the server-connection channel MCAUSER attribute is blank, and no user ID is received from the client then the user ID that started the server-connection channel is used.

The IBM MQ client does not flow the asserted user ID to the server when a client-side security exit is in use.

The user ID that runs the channel program

When the user ID fields are derived from the user ID that started the server-connection channel, the following value is used:

- z/OS For z/OS, the user ID assigned to the channel initiator started task by the z/OS started procedures table.
- For TCP/IP (non- z/OS), the user ID from the `inetd.conf` entry, or the user ID that started the listener.
- For SNA (non- z/OS), the user ID from the SNA Server entry or (if there is none) the incoming attach request, or the user ID that started the listener.
- For NetBIOS or SPX, the user ID that started the listener.

If any server-connection channel definitions exist that have the MCAUSER attribute set to blank, clients can use this channel definition to connect to the queue manager with access authority determined by the user ID supplied by the client. This might be a security exposure if the system on which the queue manager is running allows unauthorized network connections. The IBM MQ default server-connection channel (SYSTEM.DEF.SVRCONN) has the MCAUSER attribute set to blank. To prevent unauthorized access, update the MCAUSER attribute of the default definition with a user ID that has no access to IBM MQ MQ objects.

Case of user IDs

When you define a channel with `runmqsc`, the MCAUSER attribute is changed to uppercase unless the user ID is contained within single quotation marks.

ALW For servers on AIX, Linux, and Windows, the content of the `MCAUserIdentifier` field that is received from the client is changed to lowercase.

IBM i For servers on IBM i, the content of the `LongMCAUserIdentifier` field that is received from the client is changed to uppercase.

Linux AIX For servers on AIX and Linux systems, the content of the `LongMCAUserIdentifier` field that is received from the client is changed to lowercase.

By default, the user ID that is passed when an IBM MQ JMS binding application is used, is the user ID for the JVM the application is running on.

It is also possible to pass a user ID via the `createQueueConnection` method.

Planning confidentiality

Plan how to keep your data confidential.

You can implement confidentiality at the application level or at link level. You might choose to use TLS, in which case you must plan your usage of digital certificates. You can also use channel exit programs if standard facilities do not satisfy your requirements.

Related concepts

"Comparing link level security and application level security" on page 104
This topic contains information about various aspects of link level security and application level security, and compares the two levels of security.

"Channel exit programs" on page 109
Channel exit programs are programs that are called at defined places in the processing sequence of an MCA. Users and vendors can write their own channel exit programs. Some are supplied by IBM.

"Protecting channels with SSL/TLS" on page 115
TLS support in IBM MQ uses the queue manager authentication information object, and various MQSC commands. You must also consider your use of digital certificates.

Comparing link level security and application level security

This topic contains information about various aspects of link level security and application level security, and compares the two levels of security.

Link level and application level security are illustrated in Figure 10 on page 105.

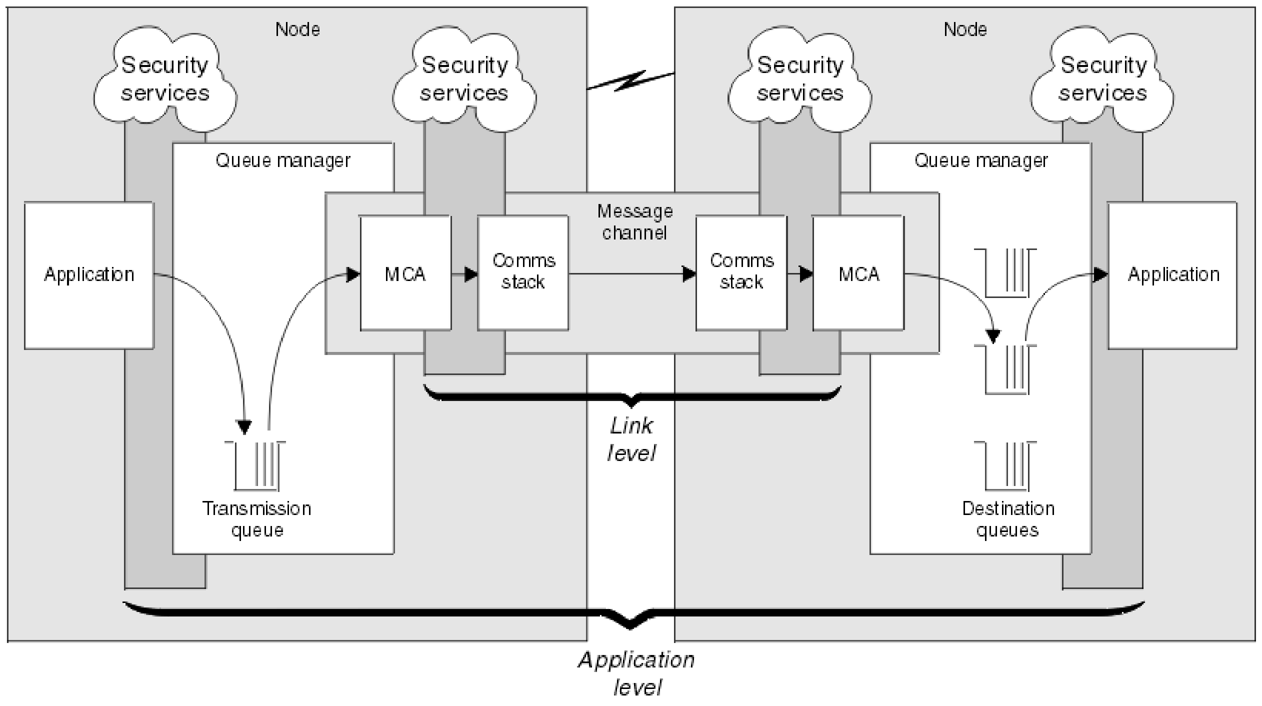


Figure 10. Link level security and application level security

Protecting messages in queues

Link level security can protect messages while they are transferred from one queue manager to another. It is particularly important when messages are transmitted over an insecure network. It cannot, however, protect messages while they are stored in queues at either a source queue manager, a destination queue manager, or an intermediate queue manager.

z/OS z/OS data set encryption can provide some protection of messages stored on queues, but only for data at rest on a local queue manager. See the section, confidentiality for data at rest on IBM MQ for z/OS with data set encryption. for more information.

Application level security, by comparison, can protect messages while they are stored in queues and applies even when distributed queuing is not used. This is the major difference between link level security and application level security and is illustrated in Figure 10 on page 105.

Queue managers not running in controlled and trusted environments

If a queue manager is running in a controlled and trusted environment, the access control mechanisms provided by IBM MQ might be considered sufficient to protect the messages stored on its queues. This is particularly true if only local queuing is involved and messages never leave the queue manager. Application level security in this case might be considered unnecessary.

Application level security might also be considered unnecessary if messages are transferred to another queue manager that is also running in a controlled and trusted environment, or are received from such a queue manager. The need for application level security becomes greater when messages are transferred to, or received from, a queue manager that is not running in a controlled and trusted environment.

Differences in cost

Application level security might cost more than link level security in terms of administration and performance.

The cost of administration is likely to be greater because there are potentially more constraints to configure and maintain. For example, you might need to ensure that a particular user sends only certain types of message and sends messages only to certain destinations. Conversely, you might need to ensure that a particular user receives only certain types of message and receives messages only from certain sources. Instead of managing the link level security services on a single message channel, you might need to be configuring and maintaining rules for every pair of users who exchange messages across that channel.

There might be an effect on performance if security services are invoked every time an application puts or gets a message.

Organizations tend to consider link level security first because it might be easier to implement. They consider application level security if they discover that link level security does not satisfy all their requirements.

Availability of components

Generally, in a distributed environment, a security service requires a component on at least two systems. For example, a message might be encrypted on one system and decrypted on another. This applies to both link level security and application level security.

In a heterogeneous environment, with different platforms in use, each with different levels of security function, the required components of a security service might not be available for every platform on which they are needed and in a form that is easy to use. This is probably more of an issue for application level security than for link level security, particularly if you intend to provide your own application level security by buying in components from various sources.

Messages in a dead letter queue

If a message is protected by application level security, there might be a problem if, for any reason, the message does not reach its destination and is put on a dead letter queue. If you cannot work out how to process the message from the information in the message descriptor and the dead letter header, you might need to inspect the contents of the application data. You cannot do this if the application data is encrypted and only the intended recipient can decrypt it.

What application level security cannot do

Application level security is not a complete solution. Even if you implement application level security, you might still require some link level security services. For example:

- When a channel starts, the mutual authentication of the two MCAs might still be a requirement. This can be done only by a link level security service.
- Application level security cannot protect the transmission queue header, MQXQH, which includes the embedded message descriptor. Nor can it protect the data in IBM MQ channel protocol flows other than message data. Only link level security can provide this protection.
- If application level security services are invoked at the server end of an MQI channel, the services cannot protect the parameters of MQI calls that are sent over the channel. In particular, the application data in an MQPUT, MQPUT1, or MQGET call is unprotected. Only link level security can provide the protection in this case.

Link level security

Link level security refers to those security services that are invoked, directly or indirectly, by an MCA, the communications subsystem, or a combination of the two working together.

Link level security is illustrated in Figure 10 on page 105.

Here are some examples of link level security services:

- The MCA at each end of a message channel can authenticate its partner. This is done when the channel starts and a communications connection has been established, but before any messages start to flow. If authentication fails at either end, the channel is closed and no messages are transferred. This is an example of an identification and authentication service.

- A message can be encrypted at the sending end of a channel and decrypted at the receiving end. This is an example of a confidentiality service.
- A message can be checked at the receiving end of a channel to determine whether its contents have been deliberately modified while it was being transmitted over the network. This is an example of a data integrity service.

Link level security provided by IBM MQ

The primary means of provision of confidentiality and data integrity in IBM MQ is by the use of TLS. For more information about the use of TLS in IBM MQ, see "TLS security protocols in IBM MQ " on page 24. For authentication, IBM MQ provides the facility to use channel authentication records. Channel authentication records offer precise control over the access granted to connecting systems, at the level of individual channels or groups of channels. For more information, see "Channel authentication records" on page 51.

Providing your own link level security

You can provide your own link level security services. Writing your own channel exit programs is the main way to provide your own link level security services.

Channel exit programs are introduced in "Channel exit programs" on page 109. The same topic also describes the channel exit program that is supplied with IBM MQ for Windows (the SSPI channel exit program). This channel exit program is supplied in source format so that you can modify the source code to suit your requirements. If this channel exit program, or channel exit programs available from other vendors, do not meet your requirements, you can design and write your own. This topic suggests ways in which channel exit programs can provide security services. For information about how to write a channel exit program, see Writing channel-exit programs.

Link level security using a security exit

Security exits normally work in pairs; one at each end of a channel. They are called immediately after the initial data negotiation has completed on channel startup.

Security exits can be used to provide identification and authentication, access control, and confidentiality.

Link level security using a message exit

A message exit can be used only on a message channel, not on an MQI channel. It has access to both the transmission queue header, MQXQH, which includes the embedded message descriptor, and the application data in a message. It can modify the contents of the message and change its length.

A message exit can be used for any purpose that requires access to the whole message rather than a portion of it.

Message exits can be used to provide identification and authentication, access control, confidentiality, data integrity, and non-repudiation, and for reasons other than security.

Link level security using send and receive exits

Send and receive exits can be used on both message and MQI channels. They are called for all types of data that flow on a channel, and for flows in both directions.

Send and receive exits have access to each transmission segment. They can modify its contents and change its length.

On a message channel, if an MCA needs to split a message and send it in more than one transmission segment, a send exit is called for each transmission segment containing a portion of the message and, at the receiving end, a receive exit is called for each transmission segment. The same occurs on an MQI channel if the input or output parameters of an MQI call are too large to be sent in a single transmission segment.

On an MQI channel, byte 10 of a transmission segment identifies the MQI call, and indicates whether the transmission segment contains the input or output parameters of the call. Send and receive exits can examine this byte to determine whether the MQI call contains application data that might need to be protected.

When a send exit is called for the first time, to acquire and initialize any resources it needs, it can ask the MCA to reserve a specified amount of space in the buffer that holds a transmission segment. When it is called later to process a transmission segment, it can use this space to add an encrypted key or a digital signature, for example. The corresponding receive exit at the other end of the channel can remove the data added by the send exit, and use it to process the transmission segment.

Send and receive exits are best suited for purposes in which they do not need to understand the structure of the data they are handling and can therefore treat each transmission segment as a binary object.

Send and receive exits can be used to provide confidentiality and data integrity, and for uses other than security.

Related tasks

Identifying the API call in a send or receive exit program

Application level security

Application level security refers to those security services that are invoked at the interface between an application and a queue manager to which it is connected.

These services are invoked when the application issues MQI calls to the queue manager. The services might be invoked, directly or indirectly, by the application, the queue manager, another product that supports IBM MQ, or a combination of any of these working together. Application level security is illustrated in Figure 10 on page 105.

Application level security is also known as *end-to-end security* or *message level security*.

Here are some examples of application level security services:

- When an application puts a message on a queue, the message descriptor contains a user ID associated with the application. However, there is no data present, such as an encrypted password, that can be used to authenticate the user ID. A security service can add this data. When the message is eventually retrieved by the receiving application, another component of the service can authenticate the user ID using the data that has travelled with the message. This is an example of an identification and authentication service.
- A message can be encrypted when it is put on a queue by an application and decrypted when it is retrieved by the receiving application. This is an example of a confidentiality service.
- A message can be checked when it is retrieved by the receiving application. This check determines whether its contents have been deliberately modified since it was first put on a queue by the sending application. This is an example of a data integrity service.

Planning for Advanced Message Security

Advanced Message Security (AMS) is a component of IBM MQ that provides a high level of protection for sensitive data flowing through the IBM MQ network, while not impacting the end applications.

If you are moving highly sensitive or valuable information, especially confidential or payment-related information such as patient records or credit card details, you must pay special attention to information security. Ensuring that information moving around the enterprise retains its integrity and is protected from unauthorized access is an ongoing challenge and responsibility. You are also likely to be required to comply with security regulations, at the risk of penalties for non-compliance.

You can develop your own security extensions to IBM MQ. However, such solutions require specialist skills and can be complicated and expensive to maintain. Advanced Message Security helps address these challenges when moving information around the enterprise between virtually every type of commercial IT system.

Advanced Message Security extends the security features of IBM MQ in the following ways:

- It provides application-level, end-to-end data protection for your point to point messaging infrastructure, using either encryption or digital signing of messages.
- It provides comprehensive security without writing complex security code or modifying or recompiling existing applications.

- It uses Public Key Infrastructure (PKI) technology to provide authentication, authorization, confidentiality, and data integrity services for messages.
- It provides administration of security policies for mainframe and distributed servers.
- It supports both IBM MQ servers and clients.
- It integrates with Managed File Transfer to provide an end-to-end secure messaging solution.

For more information, see “Advanced Message Security” on page 583.

Providing your own application level security
You can provide your own application level security services. To help you implement application level security, IBM MQ provides two exits, the API exit and the API-crossing exit.

The API exit and the API-crossing exit can provide identification and authentication, access control, confidentiality, data integrity, and non-repudiation services, and other functions not related to security.

If the API exit or API-crossing exit is not supported in your system environment, you might want to consider other ways of providing your own application level security. One way is to develop a higher level API that encapsulates the MQI. Programmers then use this API, instead of the MQI, to write IBM MQ applications.

The most common reasons for using a higher level API are:

- To hide the more advanced features of the MQI from programmers.
- To enforce standards in the use of the MQI.
- To add function to the MQI. This additional function can be security services.

Some vendor products use this technique to provide application level security for IBM MQ.

If you are planning to provide security services in this way, note the following regarding data conversion:

- If a security token, such as a digital signature, has been added to the application data in a message, any code performing data conversion must be aware of the presence of this token.
- A security token might have been derived from a binary image of the application data. Therefore, any checking of the token must be done before converting the data.
- If the application data in a message has been encrypted, it must be decrypted before data conversion.

Channel exit programs

Channel exit programs are programs that are called at defined places in the processing sequence of an MCA. Users and vendors can write their own channel exit programs. Some are supplied by IBM.

There are several types of channel exit program, but only four have a role in providing link level security:

- Security exit
- Message exit
- Send exit
- Receive exit

These four types of channel exit program are illustrated in Figure 11 on page 110 and are described in the following topics.

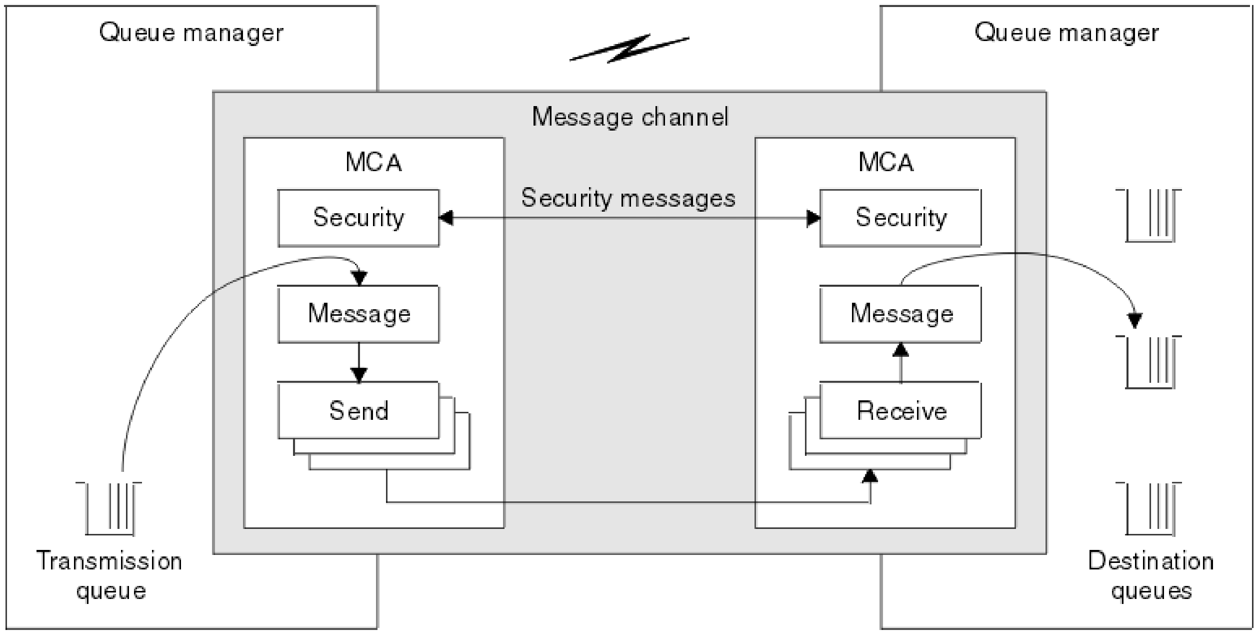


Figure 11. Security, message, send, and receive exits on a message channel

Related concepts

Channel-exit programs for messaging channels

Security exit overview

Security exits normally work in pairs. They are called before messages flow and their purpose is to allow an MCA to authenticate its partner.

Security exits normally work in pairs; one at each end of a channel. They are called immediately after the initial data negotiation has completed on channel startup, but before any messages start to flow. The primary purpose of the security exit is to enable the MCA at each end of a channel to authenticate its partner. However, there is nothing to prevent a security exit from performing other function, even function that has nothing to do with security.

Security exits can communicate with each other by sending *security messages*. The format of a security message is not defined and is determined by the user. One possible outcome of the exchange of security messages is that one of the security exits might decide not to proceed any further. In that case, the channel is closed and messages do not flow. If there is a security exit at only one end of a channel, the exit is still called and can elect whether to continue or to close the channel.

Security exits can be called on both message and MQI channels. The name of a security exit is specified as a parameter in the channel definition at each end of a channel.

For more information about security exits, see "Link level security using a security exit" on page 107.

Message exit

Message exits operate only on message channels and normally work in pairs. A message exit can operate on the whole message and make various changes to it.

Message exits at the sending and receiving ends of a channel normally work in pairs. A message exit at the sending end of a channel is called after the MCA has got a message from the transmission queue. At the receiving end of a channel, a message exit is called before the MCA puts a message on its destination queue.

A message exit has access to both the transmission queue header, MQXQH, which includes the embedded message descriptor, and the application data in a message. A message exit can modify the contents of the message and change its length. A change of length might be the result of compressing, decompressing, encrypting, or decrypting the message. It might also be the result of adding data to the message, or removing data from it.

Message exits can be used for any purpose that requires access to the whole message, rather than a portion of it, and not necessarily for security.

A message exit can determine that the message it is currently processing should not proceed any further towards its destination. The MCA then puts the message on the dead letter queue. A message exit can also close the channel.

Message exits can be called only on message channels, not on MQI channels. This is because the purpose of an MQI channel is to enable the input and output parameters of MQI calls to flow between the IBM MQ MQI client application and the queue manager.

The name of a message exit is specified as a parameter in the channel definition at each end of a channel. You can also specify a list of message exits to be run in succession.

For more information about message exits, see "Link level security using a message exit" on page 107.

Send and receive exits

Send and receive exits typically work in pairs. They operate on transmission segments and are best used where the structure of the data they are processing is not relevant.

A *send exit* at one end of a channel and a *receive exit* at the other end normally work in pairs. A send exit is called just before an MCA issues a communications send to send data over a communications connection. A receive exit is called just after an MCA has regained control following a communications receive and has received data from a communications connection. If sharing conversations is in use, over an MQI channel, a different instance of a send and receive exit is called for each conversation.

The IBM MQ channel protocol flows between two MCAs on a message channel contain control information as well as message data. Similarly, on an MQI channel, the flows contain control information as well as the parameters of MQI calls. Send and receive exits are called for all types of data.

Message data flows in only one direction on a message channel but, on an MQI channel, the input parameters of an MQI call flow in one direction and the output parameters flow in the other. On both message and MQI channels, control information flows in both directions. As a result, send and receive exits can be called at both ends of a channel.

The unit of data that is transmitted in a single flow between two MCAs is called a *transmission segment*. Send and receive exits have access to each transmission segment. They can modify its contents and change its length. A send exit, however, must not change the first 8 bytes of a transmission segment. These 8 bytes form part of the IBM MQ channel protocol header. There are also restrictions on how much a send exit can increase the length of a transmission segment. In particular, a send exit cannot increase its length beyond the maximum that was negotiated between the two MCAs at channel startup.

On a message channel, if a message is too large to be sent in a single transmission segment, the sending MCA splits the message and sends it in more than one transmission segment. As a consequence, a send exit is called for each transmission segment containing a portion of the message and, at the receiving end, a receive exit is called for each transmission segment. The receiving MCA reconstitutes the message from the transmission segments after they have been processed by the receive exit.

Similarly, on an MQI channel, the input or output parameters of an MQI call are sent in more than one transmission segment if they are too large. This might occur, for example, on an MQPUT, MQPUT1, or MQGET call if the application data is sufficiently large.

Taking these considerations into account, it is more appropriate to use send and receive exits for purposes in which they do not need to understand the structure of the data they are handling and can therefore treat each transmission segment as a binary object.

A send or a receive exit can close a channel.

The names of a send exit and a receive exit are specified as parameters in the channel definition at each end of a channel. You can also specify a list of send exits to be run in succession. Similarly, you can specify a list of receive exits.

For more information about send and receive exits, see "Link level security using send and receive exits" on page 107.

Planning data integrity

Plan how to preserve the integrity of your data.

You can implement data integrity at the application level or at link level.

At the application level, you can use API exit programs if standard facilities do not satisfy your requirements. You might choose to use Advanced Message Security (AMS) to digitally sign messages in order to protect against unauthorized modification.

At the link level, you might choose to use TLS, in which case you must plan your usage of digital certificates. You can also use channel exit programs if standard facilities do not satisfy your requirements.

Related concepts

“Protecting channels with SSL/TLS” on page 115
TLS support in IBM MQ uses the queue manager authentication information object, and various MQSC commands. You must also consider your use of digital certificates.

“Data integrity” on page 10
The *data integrity* service detects whether there has been unauthorized modification of data.

“Planning for Advanced Message Security” on page 108
Advanced Message Security (AMS) is a component of IBM MQ that provides a high level of protection for sensitive data flowing through the IBM MQ network, while not impacting the end applications.

Related reference

API exit reference

Channel-exit calls and data structures

Planning auditing

Decide what data you need to audit, and how you will capture and process audit information. Consider how to check that your system is correctly configured.

There are several aspects to activity monitoring. The aspects you must consider are often defined by auditor requirements, and these requirements are often driven by regulatory standards such as HIPAA (Health Insurance Portability and Accountability Act) or SOX (Sarbanes-Oxley). IBM MQ provides features intended to help with compliance to such standards.

Consider whether you are interested only in exceptions or whether you are interested in all system behavior.

Some aspects of auditing can also be considered as operational monitoring; one distinction for auditing is that you are often looking at historic data, not just looking at real-time alerts. Monitoring is covered in the section Monitoring and performance.

What data to audit

Consider what types of data or activity you need to audit, as described in the following sections:

Changes made to IBM MQ using the IBM MQ interfaces
Configure IBM MQ to issue instrumentation events, specifically command events and configuration events.

Changes made to IBM MQ outside its control
Some changes can affect how IBM MQ behaves, but cannot be directly monitored by IBM MQ. Examples of such changes include changes to the configuration files `mqs.ini`, `qm.ini`, and `mqclient.ini`, the creation and deletion of queue managers, installation of binary files such as user exit programs, and changes to file permissions. To monitor these activities, you must use tools running at the level of the operating system. Different tools are available and appropriate for different operating systems. You might also have logs created by associated tools such as *sudo*.

Operational control of IBM MQ
You might have to use operating system tools to audit activities such as the starting and stopping of queue managers. In some cases, IBM MQ can be configured to issue instrumentation events.

Application activity within IBM MQ
To audit the actions of applications, for example opening of queues and putting and getting of messages, configure IBM MQ to issue appropriate events.

Intruder alerts
To audit attempted breaches of security, configure your system to issue authorization events. Channel events might also be useful to show activity, particularly if a channel ends unexpectedly.

Planning the capture, display, and archiving of audit data

Many of the elements you need are reported as IBM MQ event messages. You must choose tools that can read and format these messages. If you are interested in long-term storage and analysis you must move them to an auxiliary storage mechanism such as a database. If you do not process these messages, they remain on the event queue, possibly filling the queue. You might decide to implement a tool that automatically takes action based on some events; for example, to issue an alert when a security failure happens.

Verifying that your system is correctly configured

A set of tests are supplied with the IBM MQ Explorer. Use these to check your object definitions for problems.

Also, check periodically that the system configuration is as you expect. Although command and configuration events can report when something is changed, it is also useful to dump the configuration and compare it to a known good copy.

Planning security by topology

This section covers security in specific situations, namely for channels, queue manager clusters, publish/subscribe and multicast applications, and when using a firewall.

See the following subtopics for more information:

Channel authorization

When you send or receive a message through a channel, you need to provide access to various IBM MQ resources. Message Channel Agents (MCAs) are essentially IBM MQ applications that move messages between queue managers, and as such require access to various IBM MQ resources to operate correctly.

To receive messages at PUT time for MCAs, you can use either the user ID associated with the MCA, or the user ID associated with the message.

At CONNECT time you can map the asserted user ID to an alternative user, by using **CHLAUTH** channel authentication records.

In IBM MQ, channels can be protected by TLS support.

The user IDs associated with sending and receiving channels, excluding the sender channel where the MCAUSER attribute is unused, require access to the following resources:

- The user ID associated with a sending channel requires access to the queue manager, the transmission queue, the dead-letter queue, and access to any other resources that are required by channel exits.
- The MCAUSER user ID of a receiver channel needs *+setall* authority. The reason is that the receiver channel has to create the full MQMD, including all context fields, using the data it received from the remote sender channel. The queue manager therefore requires that the user performing this activity has the *+setall* authority. This *+setall* authority must be granted to the user for:
  - All queues that the receiver channel validly puts messages to.
  - The queue manager object. For more information, see Authorizations for context.
- The MCAUSER user ID of a receiver channel where the originator requested a COA report message needs *+passid* authority on the transmission queue that returns the report message. Without this authority, AMQ8077 error messages are logged.

- With the user ID associated with the receiving channel, you can open the target queues to put messages onto the queues. This involves the Message queuing Interface (MQI), so additional access control checks might need to be made if you are not using the IBM MQ Object Authority Manager (OAM). You can specify whether the authorization checks are made against the user ID associated with the MCA (as described in this topic), or against the user ID associated with the message (from the MQMD UserIdentifier field).

  For the channel types to which it applies, the **PUTAUT** parameter of a channel definition specifies which user ID is used for these checks.

  - The channel defaults to using the queue manager's service account, which has full administrative rights and requires no special authorizations.
  - In the case of server-connection channels, administrative connections are blocked by default by CHLAUTH rules and require explicit provisioning.
  - Channels of type receiver, requester, and cluster-receiver allow local administration by any adjacent queue manager, unless the administrator takes steps to restrict this access.
  - It is not necessary to grant *dsp* and *ctrlx* authority for the MCAUSER user ID of a receiver channel.
  - Before IBM MQ 8.0.0 Fix Pack 4, if you use a user ID that lacks IBM MQ administrative privileges, then you must grant **dsp** and **ctrlx** authority for the channel to that user ID for the channel to work.

    From IBM MQ 8.0.0 Fix Pack 4, there are no authority checks when a channel resynchronizes itself and corrects sequence numbers.

    However, issuing a RESET CHANNEL command manually does still require **+dsp** and **+ctrlx** at all releases.

    Attention: When a channel reset is needed for message batch confirmation, IBM MQ attempts to query the channel, which does require **+dsp** authority.
  - The MCAUSER attribute is unused for the SDR channel type.
  - If you use the user ID associated with the message, it is likely that the user ID is from a remote system. This remote system user ID must be recognized by the target system. The following commands are examples of the type of command that you can issue to grant authority to a user ID from a remote system:

```
setmqaut -m QMgrName -t qmgr -g GroupName +connect +inq +setall
```

```
setmqaut -m QMgrName -t chl -n Profile -g GroupName +dsp +ctrlx
```

where *Profile* is a channel.

```
setmqaut -m QMgrName -t q -n Profile -g GroupName +put +setall
```

where *Profile* is a dead-letter queue, if set.

```
setmqaut -m QMgrName -t q -n Profile -g GroupName +put +setall
```

where *Profile* is a list of authorized queues.

Attention: Exercise caution when authorizing a user ID to place messages onto the Command Queue or other sensitive system queues.

The user ID associated with the MCA depends on the type of MCA. There are two types of MCA:

Caller MCA
MCAs that initiate a channel. Caller MCAs can be started as individual processes, as threads of the channel initiator, or as threads of a process pool. The user ID used is the user ID associated with the parent process (the channel initiator), or the user ID associated with the process that starts the MCA.

Responder MCA

Responder MCAs are MCAs that are started as a result of a request by a caller MCA. Responder MCAs can be started as individual processes, as threads of the listener, or as threads of a process pool. The user ID can be any one of the following types (in this order of preference):

1. On APPC, the caller MCA can indicate the user ID to be used for the responder MCA. This is called the network user ID and applies only to channels started as individual processes. Set the network user ID by using the USERID parameter of the channel definition.
2. If the **USERID** parameter is not used, the channel definition of the responder MCA can specify the user ID that the MCA must use. Set the user ID by using the **MCAUSER** parameter of the channel definition.
3. If the user ID has not been set by either of the previous (two) methods, the user ID of the process that starts the MCA or the user ID of the parent process (the listener) is used.

Related concepts

"Channel authentication records" on page 51
To exercise more precise control over the access granted to connecting systems at a channel level, you can use channel authentication records.

Related reference

Channel authentication record properties

Protecting channel initiator definitions

Only members of the mqm group can manipulate channel initiators.

IBM MQ channel initiators are not IBM MQ objects; access to them is not controlled by the OAM. IBM MQ does not allow users or applications to manipulate these objects, unless their user ID is a member of the mqm group. If you have an application that issues the PCF command **StartChannelInitiator**, the user ID specified in the message descriptor of the PCF message must be a member of the mqm group on the target queue manager.

A user ID must also be a member of the mqm group on the target machine to issue the equivalent MQSC commands through the Escape PCF command or using `runmqsc` in indirect mode.

Transmission queues

Queue managers automatically put remote messages on a transmission queue; no special authority is required for this.

However, if you need to put a message directly on a transmission queue, this requires special authorization; see Table 12 on page 132.

Channel exits

If channel authentication records are not suitable, you can use channel exits for added security. A security exit forms a secure connection between two security exit programs. One program is for the sending message channel agent (MCA), and one is for the receiving MCA.

See "Channel exit programs" on page 109 for more information about channel exits.

Protecting channels with SSL/TLS

TLS support in IBM MQ uses the queue manager authentication information object, and various MQSC commands. You must also consider your use of digital certificates.

Digital certificates and key repositories

It is good practice to set the queue manager certificate label attribute (**CERTLABL**) to the name of the personal certificate to be used for the majority of channels, and override it for exceptions, by setting the certificate label on those channels that require different certificates.

If you need many channels with certificates that differ from the default certificate set on the queue manager, you should consider dividing the channels between several queue managers or use an MQIPT proxy in front of the queue manager to present a different certificate.

You can use a different certificate for every channel, but if you store too many certificates in a key repository, you might expect performance to be affected when starting TLS channels. Try to keep the number of certificates in a key repository to less than about 50 and consider 100 to be a maximum as IBM Global Security Kit (GSKit) performance decreases sharply with larger key repositories.

Allowing multiple certificates on the same queue manager increases the chances that multiple CA certificates will be used on the same queue manager. This increases the odds of certificate Subject Distinguished Name namespace clashes for certificates issued by separate certificate authorities.

While professional certificate authorities are likely to be more careful, in-house certificate authorities often lack clear naming conventions and you could end up with unintended matches between one CA and another.

You should check the certificate Issuer Distinguished Name in addition to the Subject Distinguished Name. To do this, use a channel authentication SSLPEERMAP record and set both the **SSLPEER** and **SSLCERTI** fields to match the Subject DN and Issuer DN respectively.

Self-signed and CA-signed certificates

It is important to plan your use of digital certificates, both when you are developing and testing your application, and for its use in production. You can use CA-signed certificates or self-signed certificates, depending on the usage of your queue managers and client applications.

CA-signed certificates
For production systems, obtain your certificates from a trusted certificate authority (CA). When you obtain a certificate from an external CA, you pay for the service.

Self-signed certificates
While you are developing your application you can use self-signed certificates or certificates issued by a local CA, depending on platform:

ALW On AIX, Linux, and Windows systems, you can use self-signed certificates. See "Creating a self-signed personal certificate on AIX, Linux, and Windows" on page 529for instructions.

IBM i On IBM i systems, you can use certificates signed by the local CA. See "Requesting a server certificate on IBM i" on page 280 for instructions.

z/OS On z/OS, you can use either self-signed or local CA-signed certificates. See "Creating a self-signed personal certificate on z/OS" on page 305 or "Requesting a personal certificate on z/OS" on page 305 for instructions.

Self-signed certificates are not suitable for production use, for the following reasons:

- Self-signed certificates cannot be revoked, which might allow an attacker to spoof an identity after a private key has been compromised. CAs can revoke a compromised certificate, which prevents its further use. CA-signed certificates are therefore safer to use in a production environment, though self-signed certificates are more convenient for a test system.
- Self-signed certificates never expire. This is both convenient and safe in a test environment, but in a production environment it leaves them open to eventual security breaches. The risk is compounded by the fact that self-signed certificates cannot be revoked.
- A self-signed certificate is used both as a personal certificate and as a root (or trust anchor) CA certificate. A user with a self-signed personal certificate might be able to use it to sign other personal certificates. In general, this is not true of personal certificates issued by a CA, and represents a significant exposure.

CipherSpecs and digital certificates

Only a subset of the supported CipherSpecs can be used with all of the supported types of digital certificate. It is therefore necessary to choose an appropriate CipherSpec for your digital certificates. Similarly, if your organization's security policy requires that a particular CipherSpec be used, then you must obtain suitable digital certificates.

For more information on the relationship between CipherSpecs and digital certificates, refer to "Digital certificates and CipherSpec compatibility in IBM MQ" on page 47

Certificate validation policies

The IETF RFC 5280 standard specifies a series of certificate validation rules which compliant application software must implement in order to prevent impersonation attacks. A set of certificate validation rules is known as a certificate validation policy. For more information about certificate validation policies in IBM MQ, see "Certificate validation policies in IBM MQ" on page 45.

Planning for certificate revocation checking

Allowing multiple certificates from different certificate authorities potentially causes unnecessary additional certificate revocation checking.

In particular, if you have explicitly configured the use of a revocation server from a particular CA, for example by using an AUTHINFO object or Authentication information record (MQAIR) structure, a revocation check fails when presented with a certificate from a different CA.

You should avoid explicit certificate revocation server configuration. Instead, you should enable implicit checking where each certificate contains its own revocation server location in a certificate extension, for example, CRL Distribution Point, or OCSP AuthorityInfoAccess.

For more information, see OCSPCheckExtensions and CDPCheckExtensions.

Commands and attributes for TLS support

The Transport Layer Security (TLS) protocol provides channel security, with protection against eavesdropping, tampering, and impersonation. IBM MQ support for TLS enables you to specify, on the channel definition, that a particular channel uses TLS security. You can also specify details of the type of security you want, such as the encryption algorithm you want to use.

- The following MQSC commands support TLS:

  ALTER AUTHINFO
  Modifies the attributes of an authentication information object.

  DEFINE AUTHINFO
  Creates an authentication information object.

  DELETE AUTHINFO
  Deletes an authentication information object.

  DISPLAY AUTHINFO
  Displays the attributes for a specific authentication information object.

- The following queue manager parameters support TLS:

  CERTLABL
  Defines a personal certificate label to use.

  KEYRPWD
  On AIX, Linux, and Windows systems, defines the password IBM MQ uses to access the key repository. This field is encrypted using the password protection system.

  SSLCRLNL
  The SSLCRLNL attribute specifies a namelist of authentication information objects which are used to provide certificate revocation locations to allow enhanced TLS certificate checking.

SSLCRYP
: On AIX, Linux, and Windows systems, sets the **`SSLCryptoHardware`** queue manager attribute. This attribute is the name of the parameter string that you can use to configure the cryptographic hardware you have on your system.

SSLEV
: Determines whether a TLS event message is reported if a channel using TLS fails to establish a TLS connection.

SSLFIPS
: Specifies whether only FIPS-certified algorithms are to be used if cryptography is carried out in IBM MQ , rather than in cryptographic hardware. If cryptographic hardware is configured, the cryptographic modules provided by the hardware product are used, and these might be FIPS-certified to a particular level. This depends on the hardware product in use.

SSLKEYR
: On AIX, Linux, and Windows systems, associates a key repository with a queue manager. The GSKit enables you to use TLS security on AIX, Linux, and Windows systems.

SSLRKEYC
: The number of bytes to be sent and received within a TLS conversation before the secret key is renegotiated. The number of bytes includes control information sent by the MCA.

- The following channel parameters support TLS:

CERTLABL
: Defines a personal certificate label to use.

SSLCAUTH
: Defines whether IBM MQ requires and validates a certificate from the TLS client.

SSLCIPH
: Specifies the encryption strength and function (CipherSpec), for example TLS_RSA_WITH_AES_128_CBC_SHA. The CipherSpec must match at both ends of channel.

SSLPEER
: Specifies the distinguished name (unique identifier) of allowed partners.

This section describes the **`setmqaut`**, **`dspmqaut`**, **`dmpmqaut`**, **`rcrmqobj`**, **`rcdmqimg`**, and **`dspmqfls`** commands to support the authentication information object. It also describes the commands that can be used to manage keys and certificates on AIX, Linux, and Windows. See the following sections:

- setmqaut
- dspmqaut
- dmpmqaut
- rcrmqobj
- rcdmqimg
- dspmqfls
- "Managing keys and certificates on AIX, Linux, and Windows" on page 527

For an overview of channel security using TLS, see

- "TLS security protocols in IBM MQ " on page 24

For details of MQSC commands associated with TLS, see

- ALTER AUTHINFO
- DEFINE AUTHINFO
- DELETE AUTHINFO
- DISPLAY AUTHINFO

For details of PCF commands associated with TLS, see

- Change, Copy, and Create Authentication Information Object

- Delete Authentication Information Object
- Inquire Authentication Information Object

z/OS *IBM MQ for z/OS server connection channel*

The IBM MQ for z/OS SVRCONN channel is not secure without implementing channel authentication, or adding a security exit using TLS. SVRCONN channels do not have a security exit defined by default.

Security concerns

SVRCONN channels are not secure as initially defined, SYSTEM.DEF.SVRCONN for example. To secure a SVRCONN channel you must set up channel authentication using the SET CHLAUTH command, or install a security exit and implement TLS.

You must use a publicly available sample security exit, write a security exit yourself, or purchase a security exit.

There are several samples available that you can use as a good starting point for writing your own SVRCONN channel security exit.

In IBM MQ for z/OS, the member CSQ4BCX3 in your hlq.SCSQC37S library is a security exit sample written in the C language. Sample CSQ4BCX3 is also shipped pre-compiled in your hlq.SCSQAUTH library.

You can implement the CSQ4BCX3 sample exit by copying the compiled member hlq.SCSQAUTH(CSQ4BCX3) into a load library that is allocated to the CSQXLIB DD in your CHIN Proc. Note that the CHIN requires the load library to be set as "Program Controlled".

Alter your SVRCONN channel to set CSQ4BCX3 as the security exit.

When a client connects using that SVRCONN channel, CSQ4BCX3 will authenticate using the **`RemoteUserIdentifier`** and **`RemotePassword`** pair from MQCD or, from IBM MQ for z/OS 9.1.4, the **`CSPUserIdPtr`** and **`CSPPasswordPtr`** pair from the MQCSP. If authentication is successful it will copy **`RemoteUserIdentifier`** into **`MCAUserIdentifier`**, changing the identity context of the thread.

For Long Term Support and Continuous Delivery before IBM MQ for z/OS 9.1.4, when a client connects using that SVRCONN channel, CSQ4BCX3 will authenticate using the **`RemoteUserIdentifier`** and **`RemotePassword`** pair from MQCD. If authentication is successful it will copy **`RemoteUserIdentifier`** into **`MCAUserIdentifier`**, changing the identity context of the thread.

If you are writing an IBM MQ Java client you can use pop-ups to query the user and set MQEnvironment.userID and MQEnvironment.password. These values will be passed when the connection is made.

Now that you have a functional security exit, there is the additional concern that the userid and password are being transmitted in plain text across the network when the connection is made, as are the contents of any subsequent IBM MQ messages. You can use TLS to encrypt this initial connection information as well as the contents of any IBM MQ messages.

Example

To secure the IBM MQ Explorer SVRCONN channel SYSTEM.ADMIN.SVRCONN complete the following steps:

1. Copy hlq.SCSQAUTH(CSQ4BCX3) into a load library that is allocated to the CSQXLIB DD in the CHINIT Proc.
2. Verify that load library is Program Controlled.
3. Alter the SYSTEM ADMIN.SVRCONN to use security exit CSQ4BCX3.
4. In IBM MQ Explorer, right-click the z/OS Queue Manager name, select **Connection Details** > **Properties** > **Userid** and enter your z/OS user ID.
5. Connect to the z/OS Queue Manager by entering a password.

Additional information

For exit CSQ4BCX3 to run in a Program Controlled environment, everything loaded into the CHIN address space must be loaded from a Program Controlled library, for example, all libraries in STEPLIB and any libraries named on CSQXLIB DD. To set a load library as Program Controlled issue RACF commands. In the following example the load library name is MY.TEST.LOADLIB.

```
RALTER PROGRAM * ADDMEM('MY.TEST.LOADLIB'//NOPADCHK)
SETROPTS WHEN(PROGRAM)REFRESH
```

To alter the SVRCONN channel to implement CSQ4BCX3, issue the following IBM MQ command:

```
ALTER CHANNEL(SYSTEM ADMIN.SVRCONN) CHLTYPE(SVRCONN) SCYEXIT(CSQ4BCX3)
```

In the example above, the SVRCONN channel name being used is SYSTEM ADMIN.SVRCONN.

See “Channel exit programs” on page 109 for more information about channel exits.

Related tasks

Writing channel exit programs on z/OS

SNA LU 6.2 security services

SNA LU 6.2 offers session level cryptography, session level authentication, and conversation level authentication.

Note: This collection of topics assumes that you have a basic understanding of Systems Network Architecture (SNA). The other documentation referred to in this section contains a brief introduction to the relevant concepts and terminology. If you require a more comprehensive technical introduction to SNA, see *Systems Network Architecture Technical Overview*, GC30-3073.

SNA LU 6.2 provides three security services:

- Session level cryptography
- Session level authentication
- Conversation level authentication

For session level cryptography and session level authentication, SNA uses the *Data Encryption Standard (DES)* algorithm. The DES algorithm is a block cipher algorithm, which uses a symmetric key for encrypting and decrypting data. Both the block and the key are 8 bytes in length.

Session level cryptography
Session level cryptography encrypts and decrypts session data using the DES algorithm. It can therefore be used to provide a link level confidentiality service on SNA LU 6.2 channels.

Logical units (LUs) can provide mandatory (or required) data cryptography, selective data cryptography, or no data cryptography.

On a *mandatory cryptographic session*, an LU encrypts all outbound data request units and decrypts all inbound data request units.

On a *selective cryptographic session*, an LU encrypts only the data request units specified by the sending transaction program (TP). The sending LU signals that the data is encrypted by setting an indicator in the request header. By checking this indicator, the receiving LU can tell which request units to decrypt before passing them on to the receiving TP.

In an SNA network, IBM MQ MCAs are transaction programs. MCAs do not request encryption for any data that they send. Selective data cryptography is not an option therefore; only mandatory data cryptography or no data cryptography is possible on a session.

For information about how to implement mandatory data cryptography, see the documentation for your SNA subsystem. Refer to the same documentation for information about stronger forms of encryption that might be available for use on your platform, such as Triple DES 24-byte encryption on z/OS.

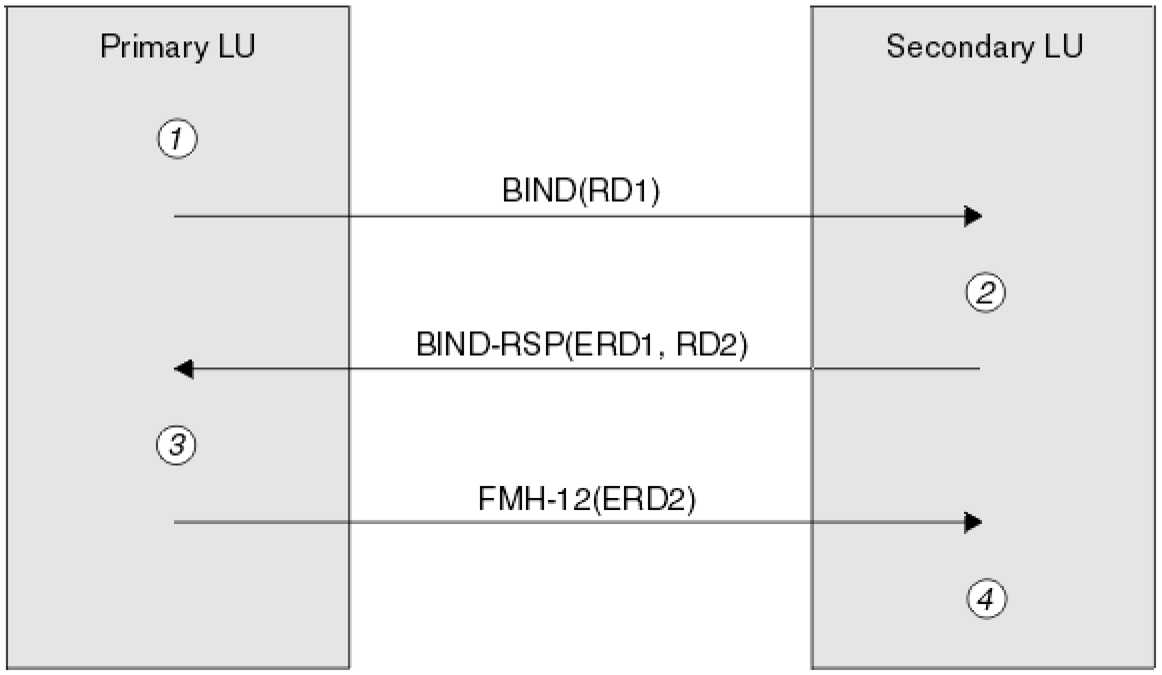
For more general information about session level cryptography, see *Systems Network Architecture LU 6.2 Reference: Peer Protocols*, SC31-6808.

Session level authentication
Session level authentication is a session level security protocol that enables two LUs to authenticate each other while they are activating a session. It is also known as *LU-LU verification*.

Because an LU is effectively the "gateway" into a system from the network, you might consider this level of authentication to be sufficient in certain circumstances. For example, if your queue manager needs to exchange messages with a remote queue manager that is running in a controlled and trusted environment, you might be prepared to trust the identities of the remaining components of the remote system after the LU has been authenticated.

Session level authentication is achieved by each LU verifying its partner's password. The password is called an *LU-LU password* because one password is established between each pair of LUs. The way that an LU-LU password is established is implementation dependent and outside the scope of SNA.

Figure 12 on page 121 illustrates the flows for session level authentication.



Legend:

BIND	=	BIND request unit
BIND-RSP	=	BIND response unit
ERD	=	Encrypted random data
FMH-12	=	Function Management Header 12
RD	=	Random data

Figure 12. Flows for session level authentication

The protocol for session level authentication is as follows. The numbers in the procedure correspond to the numbers in Figure 12 on page 121.

1. The primary LU generates a random data value (RD1) and sends it to the secondary LU in the BIND request.
2. When the secondary LU receives the BIND request with the random data, it encrypts the data using the DES algorithm with its copy of the LU-LU password as the key. The secondary LU then generates a

second random data value (RD2) and sends it, with the encrypted data (ERD1), to the primary LU in the BIND response.

3. When the primary LU receives the BIND response, it computes its own version of the encrypted data from the random data it generated originally. It does this by using the DES algorithm with its copy of the LU-LU password as the key. It then compares its version with the encrypted data that it received in the BIND response. If the two values are the same, the primary LU knows that the secondary LU has the same password as it does and the secondary LU is authenticated. If the two values do not match, the primary LU terminates the session.

   The primary LU then encrypts the random data that it received in the BIND response and sends the encrypted data (ERD2) to the secondary LU in a Function Management Header 12 (FMH-12).
4. When the secondary LU receives the FMH-12, it computes its own version of the encrypted data from the random data it generated. It then compares its version with the encrypted data that it received in the FMH-12. If the two values are the same, the primary LU is authenticated. If the two values do not match, the secondary LU terminates the session.

In an enhanced version of the protocol, which provides better protection against man in the middle attacks, the secondary LU computes a DES Message Authentication Code (MAC) from RD1, RD2, and the fully qualified name of the secondary LU, using its copy of the LU-LU password as the key. The secondary LU sends the MAC to the primary LU in the BIND response instead of ERD1.

The primary LU authenticates the secondary LU by computing its own version of the MAC, which it compares with the MAC received in the BIND response. The primary LU then computes a second MAC from RD1 and RD2, and sends the MAC to the secondary LU in the FMH-12 instead of ERD2.

The secondary LU authenticates the primary LU by computing its own version of the second MAC, which it compares with the MAC received in the FMH-12.

For information about how to configure session level authentication, see the documentation for your SNA subsystem. For more general information about session level authentication, see *Systems Network Architecture LU 6.2 Reference: Peer Protocols*, SC31-6808.

Conversation level authentication

When a local TP attempts to allocate a conversation with a partner TP, the local LU sends an attach request to the partner LU, asking it to attach the partner TP. Under certain circumstances, the attach request can contain security information, which the partner LU can use to authenticate the local TP. This is known as *conversation level authentication*, or *end user verification*.

The following topics describe how IBM MQ provides support for conversation level authentication.

For more information about conversation level authentication, see *Systems Network Architecture LU 6.2 Reference: Peer Protocols*, SC31-6808.

z/OS For information specific to z/OS, see z/OS MVS Planning: APPC/MVS Management.

For more information about CPI-C, see Using CPI Communications.

For more information about APPC/MVS TP Conversation Callable Services, see APPC/MVS TP Conversation Callable Services.

Multi *Support for conversation level authentication on Multiplatforms*

Use this topic to gain an overview of how conversation level authentication works on Multiplatforms.

The support for conversation level authentication on Multiplatforms is illustrated in Figure 13 on page 123. The numbers in the diagram correspond to the numbers in the description that follows.

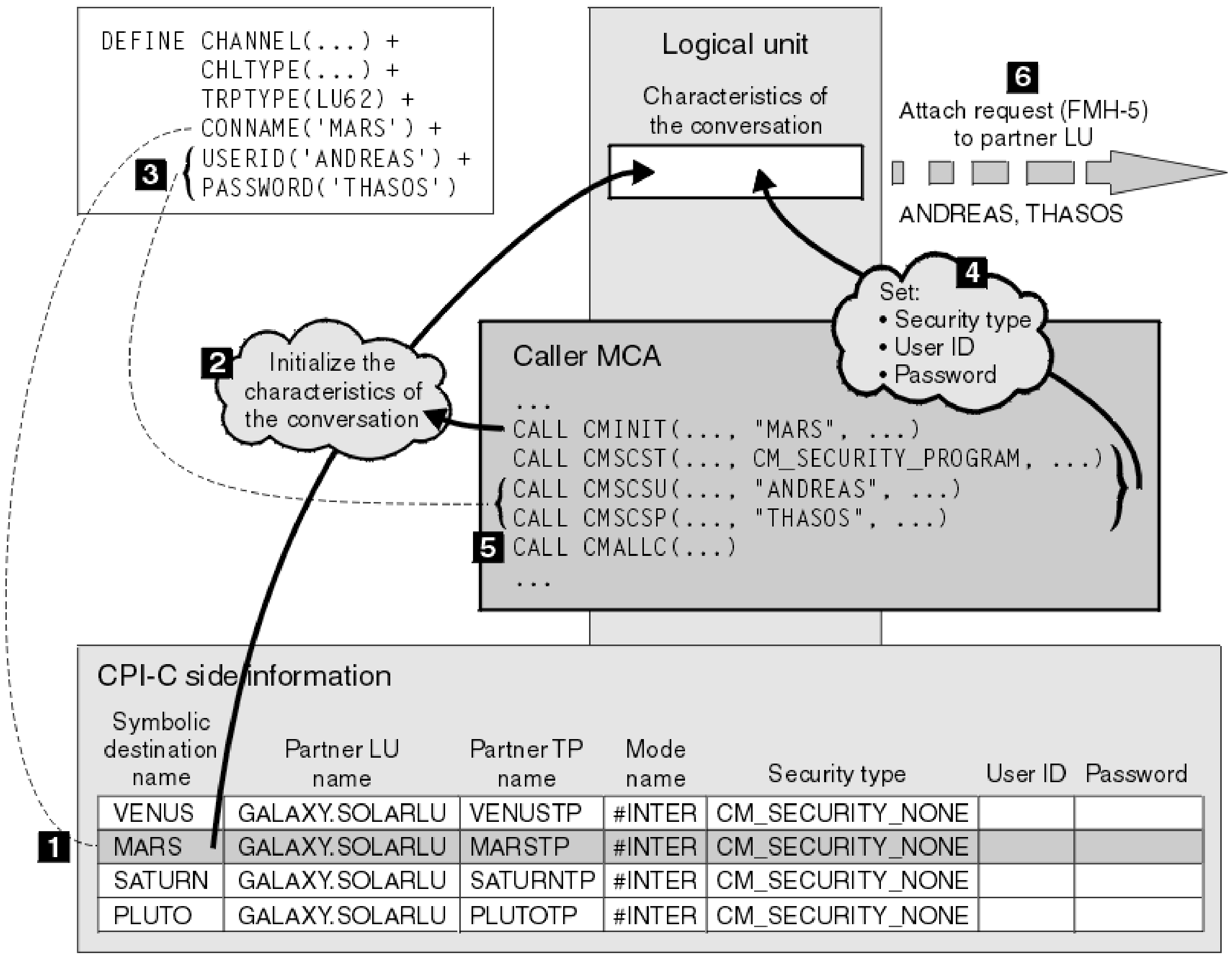

DEFINE CHANNEL(...) +
CHLTYPE(...) +
TRPTYPE(LU62) +
CONNAME('MARS') +
USERID('ANDREAS') +
PASSWORD('THASOS')
Logical unit
Characteristics of the conversation
6
Attach request (FMH-5) to partner LU
ANDREAS, THASOS
4
Set:
• Security type
• User ID
• Password
2
Initialize the characteristics of the conversation
3
Caller MCA
...
CALL CMINIT(..., "MARS", ...)
CALL CMSCST(..., CM_SECURITY_PROGRAM, ...)
CALL CMSCSU(..., "ANDREAS", ...)
CALL CMSCSP(..., "THASOS", ...)
5
CALL CMALLC(...)
...
CPI-C side information
Symbolic destination name
Partner LU name
Partner TP name
Mode name
Security type
User ID
Password
VENUS GALAXY.SOLARLU VENUSTP #INTER CM_SECURITY_NONE
1
MARS GALAXY.SOLARLU MARSTP #INTER CM_SECURITY_NONE
SATURN GALAXY.SOLARLU SATURNTP #INTER CM_SECURITY_NONE
PLUTO GALAXY.SOLARLU PLUTOTP #INTER CM_SECURITY_NONE

Figure 13. IBM MQ support for conversation level authentication

On Multiplatforms, an MCA uses Common Programming Interface Communications (CPI-C) calls to communicate with a partner MCA across an SNA network. In the channel definition at the caller end of a channel, the value of the CONNAME parameter is a symbolic destination name, which identifies a CPI-C side information entry (1). This entry specifies:

- The name of the partner LU
- The name of the partner TP, which is a responder MCA
- The name of the mode to be used for the conversation

A side information entry can also specify the following security information:

- A security type.

  The commonly implemented security types are CM_SECURITY_NONE, CM_SECURITY_PROGRAM, and CM_SECURITY_SAME, but others are defined in the CPI-C specification.
- A user ID.
- A password.

A caller MCA prepares to allocate a conversation with a responder MCA by issuing the CPI-C call CMINIT, using the value of CONNAME as one of the parameters on the call. The CMINIT call identifies, for the benefit of the local LU, the side information entry that the MCA intends to use for the conversation. The local LU uses the values in this entry to initialize the characteristics of the conversation (2).

The caller MCA then checks the values of the USERID and PASSWORD parameters in the channel definition (3). If USERID is set, the caller MCA issues the following CPI-C calls (4):

- CMSCST, to set the security type for the conversation to CM_SECURITY_PROGRAM.
- CMSCSU, to set the user ID for the conversation to the value of USERID.
- CMSCSP, to set the password for the conversation to the value of PASSWORD. CMSCSP is not called unless PASSWORD is set.

The security type, user ID, and password set by these calls override any values acquired previously from the side information entry.

The caller MCA then issues the CPI-C call CMALLC to allocate the conversation (5). In response to this call, the local LU sends an attach request (Function Management Header 5, or FMH-5) to the partner LU (6).

If the partner LU will accept a user ID and a password, the values of USERID and PASSWORD are included in the attach request. If the partner LU will not accept a user ID and a password, the values are not included in the attach request. The local LU discovers whether the partner LU will accept a user ID and a password as part of an exchange of information when the LUs bind to form a session.

In a later version of the attach request, a password substitute can flow between the LUs instead of a clear password. A password substitute is a DES Message Authentication Code (MAC), or an SHA-1 message digest, formed from the password. Password substitutes can be used only if both LUs support them.

When the partner LU receives an incoming attach request containing a user ID and a password, it might use the user ID and password for the purposes of identification and authentication. By referring to access control lists, the partner LU might also determine whether the user ID has the authority to allocate a conversation and attach the responder MCA.

In addition, the responder MCA might run under the user ID included in the attach request. In this case, the user ID becomes the default user ID for the responder MCA and is used for authority checks when the MCA attempts to connect to the queue manager. It might also be used for authority checks subsequently when the MCA attempts to access the queue manager's resources.

The way in which a user ID and a password in an attach request can be used for identification, authentication, and access control is implementation dependent. For information specific to your SNA subsystem, refer to the appropriate documentation.

If USERID is not set, the caller MCA does not call CMSCST, CMSCSU, and CMSCSP. In this case, the security information that flows in an attach request is determined solely by what is specified in the side information entry and what the partner LU will accept.

Conversation level authentication and IBM MQ for z/OS
Use this topic to gain an overview of how conversation level authentication works, on z/OS.

On IBM MQ for z/OS, MCAs do not use CPI-C. Instead, they use APPC/MVS TP Conversation Callable Services, an implementation of Advanced Program-to-Program Communication (APPC), which has some CPI-C features. When a caller MCA allocates a conversation, a security type of SAME is specified on the call. Therefore, because an APPC/MVS LU supports persistent verification only for inbound conversations, not for outbound conversations, there are two possibilities:

- If the partner LU trusts the APPC/MVS LU and will accept an already verified user ID, the APPC/MVS LU sends an attach request containing:
  - The channel initiator address space user ID
  - A security profile name, which, if RACF is used, is the name of the current connect group of the channel initiator address space user ID
  - An already verified indicator
- If the partner LU does not trust the APPC/MVS LU and will not accept an already verified user ID, the APPC/MVS LU sends an attach request containing no security information.

On IBM MQ for z/OS, the USERID and PASSWORD parameters on the DEFINE CHANNEL command cannot be used for a message channel and are valid only at the client connection end of an MQI channel. Therefore, an attach request from an APPC/MVS LU never contains values specified by these parameters.

Security for queue manager clusters

Though queue manager clusters can be convenient to use, you must pay special attention to their security.

A *queue manager cluster* is a network of queue managers that are logically associated in some way. A queue manager that is a member of a cluster is called a *cluster queue manager*.

A queue that belongs to a cluster queue manager can be made known to other queue managers in the cluster. Such a queue is called a *cluster queue*. Any queue manager in a cluster can send messages to cluster queues without needing any of the following:

- An explicit remote queue definition for each cluster queue
- Explicitly defined channels to and from each remote queue manager
- A separate transmission queue for each outbound channel

You can create a cluster in which two or more queue managers are clones. This means that they have instances of the same local queues, including any local queues declared as cluster queues, and can support instances of the same server applications.

When an application connected to a cluster queue manager sends a message to a cluster queue that has an instance on each of the cloned queue managers, IBM MQ decides which queue manager to send it to. When many applications send messages to the cluster queue, IBM MQ balances the workload across each of the queue managers that have an instance of the queue. If one of the systems hosting a cloned queue manager fails, IBM MQ continues to balance the workload across the remaining queue managers until the system that failed is restarted.

If you are using queue manager clusters, you need to consider the following security issues:

- Allowing only selected queue managers to send messages to your queue manager
- Allowing only selected users of a remote queue manager to send messages to a queue on your queue manager
- Allowing applications connected to your queue manager to send messages only to selected remote queues

These considerations are relevant even if you are not using clusters, but they become more important if you are using clusters.

If an application can send messages to one cluster queue, it can send messages to any other cluster queue without needing additional remote queue definitions, transmission queues, or channels. It therefore becomes more important to consider whether you need to restrict access to the cluster queues on your queue manager, and to restrict the cluster queues to which your applications can send messages.

There are some additional security considerations, which are relevant only if you are using queue manager clusters:

- Allowing only selected queue managers to join a cluster
- Forcing unwanted queue managers to leave a cluster

For more information about all these considerations, see Keeping clusters secure. z/OS For considerations specific to IBM MQ for z/OS, see "Security in queue manager clusters on z/OS" on page 258.

Related tasks

"Preventing queue managers receiving messages" on page 470

You can prevent a cluster queue manager from receiving messages it is unauthorized to receive by using exit programs.

Security for IBM MQ Publish/Subscribe

There are additional security considerations if you are using IBM MQ Publish/Subscribe.

In a publish/subscribe system, there are two types of application: publisher and subscriber. *Publishers* supply information in the form of IBM MQ messages. When a publisher publishes a message, it specifies a *topic*, which identifies the subject of the information inside the message.

Subscribers are the consumers of the information that is published. A subscriber specifies the topics it is interested in by subscribing to them.

The *queue manager* is an application supplied with IBM MQ Publish/Subscribe. It receives published messages from publishers and subscription requests from subscribers, and routes the published messages to the subscribers. A subscriber is sent messages only on those topics to which it has subscribed.

For more information, see Publish/subscribe security.

Multicast security

Use this information to understand why security processes might be needed with IBM MQ Multicast.

IBM MQ Multicast does not have in-built security. Security checks are handled in the queue manager at MQOPEN time and the MQMD field setting is handled by the client. Some applications in the network might not be IBM MQ applications (For example, LLM applications, see Multicast interoperability with IBM MQ Low Latency Messaging for more information), therefore you might need to implement your own security procedures because receiving applications cannot be certain of the validity of context fields.

There are three security processes to consider:

Access control

Access control in IBM MQ is based on user IDs. For more information on this subject, see "Access control for clients" on page 102.

Network security

An isolated network might be a viable security option to prevent fake messages. It is possible for an application on the multicast group address to publish malicious messages using native communication functions, which are indistinguishable from MQ messages because they come from an application on the same multicast group address.

It is also possible for a client on the multicast group address to receive messages that were intended for other clients on the same multicast group address.

Isolating the multicast network ensures that only valid clients and applications have access. This security precaution can prevent malicious messages from coming in, and confidential information from going out.

For information about multicast group network addresses, see: Setting the appropriate network for multicast traffic

Digital signatures

A digital signature is formed by encrypting a representation of a message. The encryption uses the private key of the signatory and, for efficiency, usually operates on a message digest rather than the message itself. Digitally signing a message before an MQPUT is a good security precaution, but this process might have a detrimental effect on performance if there is a large volume of messages.

Digital signatures vary with the data being signed. If two different messages are signed digitally by the same entity, the two signatures differ, but both signatures can be verified with the same public key, that is, the public key of the entity that signed the messages.

As mentioned previously in this section, it might be possible for an application on the multicast group address to publish malicious messages using native communication functions, which are indistinguishable from MQ messages. Digital signatures provide proof of origin, and only the sender knows the private key, which provides strong evidence that the sender is the originator of the message.

For more information on this subject, see "Cryptographic concepts" on page 11.

Firewalls and IBM MQ Internet Pass-Thru

IBM MQ Internet Pass-Thru can simplify communication through a firewall.

MQIPT enables two queue managers to exchange messages, or an IBM MQ client application to connect to a queue manager, without requiring a direct TCP/IP connection. This architecture is useful if a firewall prohibits a direct TCP/IP connection between two systems. Using MQIPT as a proxy can make the passage of IBM MQ channel data through a firewall simpler and more manageable. MQIPT can also protect IBM MQ data that is sent over the internet by using Transport Layer Security (TLS), and tunnel IBM MQ data inside HTTP.

For more information, see IBM MQ Internet Pass-Thru.

z/OS IBM MQ for z/OS security implementation checklist

This topic gives a step-by-step procedure you can use to work out and define the security implementation for each of your IBM MQ queue managers.

RACF provides definitions for the IBM MQ security classes in its supplied static Class Descriptor Table (CDT). As you work through the checklist, you can determine which of these classes your setup requires. You must ensure that they are activated as described in "RACF security classes" on page 184.

Refer to other sections for details, in particular "Profiles used to control access to IBM MQ resources" on page 194.

If you require security checking, follow this checklist to implement it:

1. Activate the RACF MQADMIN (uppercase profiles) or MXADMIN (mixed case profiles) class.
   - Do you want security at queue sharing group level, queue manager level, or a combination of both?

     See, "Profiles to control queue sharing group or queue manager level security" on page 189.
2. Do you need connection security?
   - **Yes**: Activate the MQCONN class. Define appropriate connection profiles at either queue manager level or queue sharing group level in the MQCONN class. Then permit the appropriate users or groups access to these profiles.

     Note: Only users of the `MQCONN` API request or CICS or IMS address space user IDs need to have access to the corresponding connection profile.
   - **No**: Define an hlq.NO.CONNECT.CHECKS profile at either queue manager level or queue sharing group level in the MQADMIN or MXADMIN class.
3. Do you need security checking on commands?
   - **Yes**: Activate the MQCMDS class. Define appropriate command profiles at either queue manager level or queue sharing group level in the MQCMDS class. Then permit the appropriate users or groups access to these profiles.

     If you are using a queue sharing group, you might need to include the user IDs used by the queue manager itself and the channel initiator. See "Setting up IBM MQ for z/OS resource security" on page 250.
   - **No**: Define an hlq.NO.CMD.CHECKS profile for the required queue manager or queue sharing group in the MQADMIN or MXADMIN class.
4. Do you need security on the resources used in commands?

- **Yes**: Ensure the MQADMIN or MXADMIN class is active. Define appropriate profiles for protecting resources on commands at either queue manager level or queue sharing group level in the MQADMIN or MXADMIN class. Then permit the appropriate users or groups access to these profiles. Set the CMDUSER parameter in CSQ6SYSP to the default user ID to be used for command security checks.

  If you are using a queue sharing group, you might need to include the user IDs used by the queue manager itself and the channel initiator. See "Setting up IBM MQ for z/OS resource security" on page 250.
- **No**: Define an hlq.NO.CMD.RESC.CHECKS profile for the required queue manager or queue sharing group in the MQADMIN or MXADMIN class.

5. Do you need queue security?
   - **Yes**: Activate the MQQUEUE or MXQUEUE class. Define appropriate queue profiles for the required queue manager or queue sharing group in the MQQUEUE or MXQUEUEclass. Then permit the appropriate users or groups access to these profiles.
   - **No**: Define an hlq.NO.QUEUE.CHECKS profile for the required queue manager or queue sharing group in the MQADMIN or MXADMIN class.
6. Do you need process security?
   - **Yes**: Activate the MQPROC or MXPROC class. Define appropriate process profiles at either queue manager or queue sharing group level and permit the appropriate users or groups access to these profiles.
   - **No**: Define an hlq.NO.PROCESS.CHECKS profile for the appropriate queue manager or queue sharing group in the MQADMIN or MXADMIN class.
7. Do you need namelist security?
   - **Yes**: Activate the MQNLIST or MXNLISTclass. Define appropriate namelist profiles at either queue manager level or queue sharing group level in the MQNLIST or MXNLIST class. Then permit the appropriate users or groups access to these profiles.
   - **No**: Define an hlq.NO.NLIST.CHECKS profile for the required queue manager or queue sharing group in the MQADMIN or MXADMIN class.
8. Do you need topic security?
   - **Yes**: Activate the MXTOPIC class. Define appropriate topic profiles at either queue manager level or queue sharing group level in the MXTOPIC class. Then permit the appropriate users or groups access to these profiles.
   - **No**: Define an hlq.NO.TOPIC.CHECKS profile for the required queue manager or queue sharing group in the MQADMIN or MXADMIN class.
9. Do any users need to protect the use of the `MQOPEN` or `MQPUT1` options relating to the use of context?
   - **Yes**: Ensure the MQADMIN or MXADMIN class is active. Define hlq.CONTEXT.queuename profiles at the queue, queue manager, or queue sharing group level in the MQADMIN or MXADMIN class. Then permit the appropriate users or groups access to these profiles.
   - **No**: Define an hlq.NO.CONTEXT.CHECKS profile for the required queue manager or queue sharing group in the MQADMIN or MXADMIN class.
10. Do you need to protect the use of alternative user IDs?
    - **Yes**: Ensure the MQADMIN or MXADMIN class is active. Define the appropriate hlq.ALTERNATE.USER. *`alternateuserid`* profiles for the required queue manager or queue sharing group and permit the required users or groups access to these profiles.
    - **No**: Define the profile hlq.NO.ALTERNATE.USER.CHECKS for the required queue manager or queue sharing group in the MQADMIN or MXADMIN class.
11. Do you need to tailor which user IDs are to be used for resource security checks through RESLEVEL?

- **Yes**: Ensure the MQADMIN or MXADMIN class is active. Define an hlq.RESLEVEL profile at either queue manager level or queue sharing group level in the MQADMIN or MXADMIN class. Then permit the required users or groups access to the profile.
- **No**: Ensure that no generic profiles exist in the MQADMIN or MXADMIN class that can apply to hlq.RESLEVEL. Define an hlq.RESLEVEL profile for the required queue manager or queue sharing group and ensure that no users or groups have access to it.

12. Do you need to 'timeout' unused user IDs from IBM MQ ?

    - **Yes**: Determine what timeout values you would like to use and issue the MQSC ALTER SECURITY command to change the TIMEOUT and INTERVAL parameters.
    - **No**: Issue the MQSC ALTER SECURITY command to set the INTERVAL value to zero.

    Note: Update the CSQINP1 initialization input data set used by your subsystem so that the MQSC ALTER SECURITY command is issued automatically when the queue manager is started.
13. Do you use distributed queuing?

    - **Yes**: Use channel authentication records. For more information, see "Channel authentication records" on page 51.
    - You can also determine the appropriate MCAUSER attribute value for each channel, or provide suitable channel security exits.
14. Do you want to use Transport Layer Security (TLS)?

    - **Yes**: To specify that any user presenting an TLS personal certificate containing a specified DN is to use a specific MCAUSER, set a channel authentication record of type SSLPEERMAP. You can specify a single distinguished name or a pattern including wildcards.
    - Plan your TLS infrastructure. Install the System SSL feature of z/OS. In RACF, set up your certificate name filters (CNFs), if you are using them, and your digital certificates. Set up your SSL key ring. Ensure that the SSLKEYR queue manager attribute is nonblank and points to your SSL key ring. Also ensure that the value of SSLTASKS is at least 2.
    - **No**: Ensure that SSLKEYR is blank, and SSLTASKS is zero.

    For further details about TLS, see "TLS security protocols in IBM MQ " on page 24.
15. Do you use clients?

    - **Yes**: Use channel authentication records.
    - You can also determine the appropriate MCAUSER attribute value for each server-connection channel, or provide suitable channel security exits if required.
16. Check your switch settings.

    IBM MQ issues messages when the queue manager is started that display your security settings. Use these messages to determine whether your switches are set correctly.
17. Do you send passwords from client applications?

    - **Yes**: Ensure that the z/OS feature is installed and Integrated Cryptographic Service Facility (ICSF) is started for the best protection.
    - **No**: You can ignore the error message reporting that ICSF has not started.

    For further information about ICSF see "Using the Integrated Cryptographic Service Facility (ICSF)" on page 258

Setting up security

This collection of topics contains information specific to different operating systems, and to the use of clients.

ALW Setting up security on AIX, Linux, and Windows

Security considerations specific to AIX, Linux, and Windows systems.

IBM MQ queue managers transfer information that is potentially valuable, so you need to use an authority system to ensure that unauthorized users cannot access your queue managers. Consider the following types of security controls:

Who can administer IBM MQ
: You can define the set of users who can issue commands to administer IBM MQ.

Who can use IBM MQ objects
: You can define which users (usually applications) can use MQI calls and PCF commands to do the following:

  - Who can connect to a queue manager.
  - Who can access objects (queues, process definitions, namelists, channels, client connection channels, listeners, services, and authentication information objects), and what type of access they have to those objects.
  - Who can access IBM MQ messages.
  - Who can access the context information associated with a message.

Channel security
: You need to ensure that channels used to send messages to remote systems can access the required resources.

You can use standard operating facilities to grant access to program libraries, MQI link libraries, and commands. However, the directory containing queues and other queue manager data is private to IBM MQ; do not use standard operating system commands to grant or revoke authorizations to MQI resources.

ALW How authorizations work on AIX, Linux, and Windows

The authorization specification tables in the topics in this section define precisely how the authorizations work and the restrictions that apply.

The tables apply to these situations:

- Applications that issue MQI calls
- Administration programs that issue MQSC commands as escape PCFs
- Administration programs that issue PCF commands

In this section, the information is presented as a set of tables that specify the following:

Action to be performed
: MQI option, MQSC command, or PCF command.

Access control object
: Queue, process, queue manager, namelist, authentication information, channel, client connection channel, listener, or service.

Authorization required
: Expressed as an MQZAO_ constant.

In the tables, the constants prefixed by MQZAO_ correspond to the keywords in the authorization list for the `setmqaut` command for the particular entity. For example, MQZAO_BROWSE corresponds to the keyword `+browse`, MQZAO_SET_ALL_CONTEXT corresponds to the keyword `+setall`, and so on. These constants are defined in the header file cmqzc.h, supplied with the product.

ALW *Authorizations for MQI calls*

MQCONN, **MQOPEN**, **MQPUT1**, and **MQCLOSE** might require authorization checks. The tables in this topic summarize the authorizations needed for each call.

An application is allowed to issue specific MQI calls and options only if the user identifier under which it is running (or whose authorizations it is able to assume) has been granted the relevant authorization.

Four MQI calls might require authorization checks: **MQCONN**, **MQOPEN**, **MQPUT1**, and **MQCLOSE**.

For **MQOPEN** and **MQPUT1**, the authority check is made on the name of the object being opened, and not on the name, or names, resulting after a name has been resolved. For example, an application might be granted authority to open an alias queue without having authority to open the base queue to which the alias resolves. The rule is that the check is carried out on the first definition encountered during the process of resolving a name that is not a queue manager alias, unless the queue manager alias definition is opened directly; that is, its name is displayed in the *ObjectName* field of the object descriptor. Authority is always needed for the object being opened. In some cases additional queue-independent authority, obtained through an authorization for the queue manager object, is required.

Table 10 on page 131, Table 11 on page 131, Table 12 on page 132, and Table 13 on page 133 summarize the authorizations needed for each call. In the tables *Not applicable* means that authorization checking is not relevant to this operation; *No check* means that no authorization checking is performed.

Note: You will find no mention of namelists, channels, client connection channels, listeners, services, or authentication information objects in these tables. This is because none of the authorizations apply to these objects, except for MQOO_INQUIRE, for which the same authorizations apply as for the other objects.

The special authorization MQZAO_ALL_MQI includes all the authorizations in the tables that are relevant to the object type, except MQZAO_DELETE and MQZAO_DISPLAY, which are classed as administration authorizations.

In order to modify any of the message context options, you must have the appropriate authorizations to issue the call. For example, in order to use MQOO_SET_IDENTITY_CONTEXT or MQPMO_SET_IDENTITY_CONTEXT, you must have +setid permission.

Table 10. Security authorization needed for MQCONN calls

Authorization required for:	**Queue object (“1” on page 133)**	**Process object**	**Queue manager object**
MQCONN	Not applicable	Not applicable	MQZAO_CONNECT

Table 11. Security authorization needed for MQOPEN calls

Authorization required for:	**Queue object (“1” on page 133)**	**Process object**	**Queue manager object**
MQOO_INQUIRE	MQZAO_INQUIRE	MQZAO_INQUIRE	MQZAO_INQUIRE
MQOO_BROWSE	MQZAO_BROWSE	Not applicable	No check
MQOO_INPUT_*	MQZAO_INPUT	Not applicable	No check
MQOO_SAVE_ ALL_CONTEXT **(“2” on page 133)**	MQZAO_INPUT	Not applicable	Not applicable
MQOO_OUTPUT (Normal queue) **(“3” on page 133)**	MQZAO_OUTPUT	Not applicable	Not applicable
MQOO_PASS_ IDENTITY_CONTEXT **(“4” on page 133)**	MQZAO_PASS_ IDENTITY_CONTEXT	Not applicable	No check

Table 11. Security authorization needed for MQOPEN calls (continued)

Authorization required for:	**Queue object (“1” on page 133)**	**Process object**	**Queue manager object**
MQOO_PASS_ALL_CONTEXT (**“4” on page 133, “5” on page 133**)	MQZAO_PASS_ALL_CONTEXT	Not applicable	No check
MQOO_SET_IDENTITY_CONTEXT (**“4” on page 133, “5” on page 133**)	MQZAO_SET_IDENTITY_CONTEXT	Not applicable	MQZAO_SET_IDENTITY_CONTEXT (**“6” on page 133**)
MQOO_SET_ALL_CONTEXT (**“4” on page 133, “7” on page 133**)	MQZAO_SET_ALL_CONTEXT	Not applicable	MQZAO_SET_ALL_CONTEXT (**“6” on page 133**)
MQOO_OUTPUT (Transmission queue) (**“8” on page 133**)	MQZAO_SET_ALL_CONTEXT	Not applicable	MQZAO_SET_ALL_CONTEXT (**“6” on page 133**)
MQOO_SET	MQZAO_SET	Not applicable	No check
MQOO_ALTERNATE_USER_AUTHORITY	(**“9” on page 133**)	(**“9” on page 133**)	MQZAO_ALTERNATE_USER_AUTHORITY (**“9” on page 133, “10” on page 133**)

Table 12. Security authorization needed for MQPUT1 calls

Authorization required for:	**Queue object (“1” on page 133)**	**Process object**	**Queue manager object**
MQPMO_PASS_IDENTITY_CONTEXT	MQZAO_PASS_IDENTITY_CONTEXT (**“11” on page 133**)	Not applicable	No check
MQPMO_PASS_ALL_CONTEXT	MQZAO_PASS_ALL_CONTEXT (**“11” on page 133**)	Not applicable	No check
MQPMO_SET_IDENTITY_CONTEXT	MQZAO_SET_IDENTITY_CONTEXT (**“11” on page 133**)	Not applicable	MQZAO_SET_IDENTITY_CONTEXT (**“6” on page 133**)
MQPMO_SET_ALL_CONTEXT	MQZAO_SET_ALL_CONTEXT (**“11” on page 133**)	Not applicable	MQZAO_SET_ALL_CONTEXT (**“6” on page 133**)
(Transmission queue) (**“8” on page 133**)	MQZAO_SET_ALL_CONTEXT	Not applicable	MQZAO_SET_ALL_CONTEXT (**“6” on page 133**)
MQPMO_ALTERNATE_USER_AUTHORITY	(**“12” on page 133**)	Not applicable	MQZAO_ALTERNATE_USER_AUTHORITY (**“10” on page 133**)

Table 13. Security authorization needed for MQCLOSE calls

Authorization required for:	**Queue object (“1” on page 133)**	**Process object**	**Queue manager object**
MQCO_DELETE	MQZAO_DELETE **(“13” on page 133)**	Not applicable	Not applicable
MQCO_DELETE _PURGE	MQZAO_DELETE **(“13” on page 133)**	Not applicable	Not applicable

Notes for the tables:

1. If opening a model queue:
   - MQZAO_DISPLAY authority is needed for the model queue, in addition to the authority to open the model queue for the type of access for which you are opening.
   - MQZAO_CREATE authority is not needed to create the dynamic queue.
   - The user identifier used to open the model queue is automatically granted all the queue-specific authorities (equivalent to MQZAO_ALL) for the dynamic queue created.
2. MQOO_INPUT_* must also be specified. This is valid for a local, model, or alias queue.
3. This check is performed for all output cases, except transmission queues (see note “8” on page 133).
4. MQOO_OUTPUT must also be specified.
5. MQOO_PASS_IDENTITY_CONTEXT is also implied by this option.
6. This authority is required for both the queue manager object and the particular queue.
7. MQOO_PASS_IDENTITY_CONTEXT, MQOO_PASS_ALL_CONTEXT, and MQOO_SET_IDENTITY_CONTEXT are also implied by this option.
8. This check is performed for a local or model queue that has a *Usage* queue attribute of MQUS_TRANSMISSION, and is being opened directly for output. It does not apply if a remote queue is being opened (either by specifying the names of the remote queue manager and remote queue, or by specifying the name of a local definition of the remote queue).
9. At least one of MQOO_INQUIRE (for any object type), or MQOO_BROWSE, MQOO_INPUT_*, MQOO_OUTPUT, or MQOO_SET (for queues) must also be specified. The check carried out is as for the other options specified, using the supplied alternate-user identifier for the specific-named object authority, and the current application authority for the MQZAO_ALTERNATE_USER_IDENTIFIER check.
10. This authorization allows any *AlternateUserId* to be specified.
11. An MQZAO_OUTPUT check is also carried out if the queue does not have a *Usage* queue attribute of MQUS_TRANSMISSION.
12. The check carried out is as for the other options specified, using the supplied alternate-user identifier for the specific-named queue authority, and the current application authority for the MQZAO_ALTERNATE_USER_IDENTIFIER check.
13. The check is carried out only if both of the following statements are true:
    - A permanent dynamic queue is being closed and deleted.
    - The queue was not created by the MQOPEN call that returned the object handle being used.

    Otherwise, there is no check.

ALW Authorizations for MQSC commands in escape PCFs

This information summarizes the authorizations needed for each MQSC command contained in Escape PCF.

Not applicable means that this operation is not relevant to this object type.

The user ID under which the program that submits the command is running must also have the following authorities:

- MQZAO_CONNECT authority to the queue manager
- MQZAO_DISPLAY authority on the queue manager in order to perform PCF commands
- Authority to issue the MQSC command within the text of the Escape PCF command

ALTER *object*

Object	**Authorization required**
Queue	MQZAO_CHANGE
Topic	MQZAO_CHANGE
Process	MQZAO_CHANGE
Queue manager	MQZAO_CHANGE
Namelist	MQZAO_CHANGE
Authentication information	MQZAO_CHANGE
Channel	MQZAO_CHANGE
Client connection channel	MQZAO_CHANGE
Listener	MQZAO_CHANGE
Service	MQZAO_CHANGE
Communication information	MQZAO_CHANGE

CLEAR *object*

Object	**Authorization required**
Queue	MQZAO_CLEAR
Topic	MQZAO_CLEAR
Process	Not applicable
Queue manager	Not applicable
Namelist	Not applicable
Authentication information	Not applicable
Channel	Not applicable
Client connection channel	Not applicable
Listener	Not applicable
Service	Not applicable
Communication information	Not applicable

DEFINE *object* NOREPLACE (“1” on page 138)

Object	**Authorization required**
Queue	MQZAO_CREATE **(“2” on page 138)**
Topic	MQZAO_CREATE **(“2” on page 138)**
Process	MQZAO_CREATE **(“2” on page 138)**

Object	**Authorization required**
Queue manager	Not applicable
Namelist	MQZAO_CREATE **(“2” on page 138)**
Authentication information	MQZAO_CREATE **(“2” on page 138)**
Channel	MQZAO_CREATE **(“2” on page 138)**
Client connection channel	MQZAO_CREATE **(“2” on page 138)**
Listener	MQZAO_CREATE **(“2” on page 138)**
Service	MQZAO_CREATE **(“2” on page 138)**
Communication information	MQZAO_CREATE **(“2” on page 138)**

DEFINE *object* REPLACE (“1” on page 138, “3” on page 138)

Object	**Authorization required**
Queue	MQZAO_CHANGE
Topic	MQZAO_CHANGE
Process	MQZAO_CHANGE
Queue manager	Not applicable
Namelist	MQZAO_CHANGE
Authentication information	MQZAO_CHANGE
Channel	MQZAO_CHANGE
Client connection channel	MQZAO_CHANGE
Listener	MQZAO_CHANGE
Service	MQZAO_CHANGE
Communication information	MQZAO_CHANGE

DELETE *object*

Object	**Authorization required**
Queue	MQZAO_DELETE
Topic	MQZAO_DELETE
Process	MQZAO_DELETE
Queue manager	Not applicable
Namelist	MQZAO_DELETE
Authentication information	MQZAO_DELETE
Channel	MQZAO_DELETE
Client connection channel	MQZAO_DELETE
Listener	MQZAO_DELETE
Service	MQZAO_DELETE
Communication information	MQZAO_DELETE

DISPLAY *object*

Object	**Authorization required**
Queue	MQZAO_DISPLAY
Topic	MQZAO_DISPLAY
Process	MQZAO_DISPLAY
Queue manager	MQZAO_DISPLAY
Namelist	MQZAO_DISPLAY
Authentication information	MQZAO_DISPLAY
Channel	MQZAO_DISPLAY
Client connection channel	MQZAO_DISPLAY
Listener	MQZAO_DISPLAY
Service	MQZAO_DISPLAY
Communication information	MQZAO_DISPLAY

START *object*

Object	**Authorization required**
Queue	Not applicable
Topic	Not applicable
Process	Not applicable
Queue manager	Not applicable
Namelist	Not applicable
Authentication information	Not applicable
Channel	MQZAO_CONTROL
Client connection channel	Not applicable
Listener	MQZAO_CONTROL
Service	MQZAO_CONTROL
Communication information	Not applicable

STOP *object*

Object	**Authorization required**
Queue	Not applicable
Topic	Not applicable
Process	Not applicable
Queue manager	Not applicable
Namelist	Not applicable
Authentication information	Not applicable
Channel	MQZAO_CONTROL

Object	Authorization required
Client connection channel	Not applicable
Listener	MQZAO_CONTROL
Service	MQZAO_CONTROL
Communication information	Not applicable

Channel Commands

Command	Object	Authorization required
PING CHANNEL	Channel	MQZAO_CONTROL
RESET CHANNEL	Channel	MQZAO_CONTROL_EXTENDED
RESOLVE CHANNEL	Channel	MQZAO_CONTROL_EXTENDED

Subscription Commands

Command	Object	Authorization required
ALTER SUB	Topic	MQZAO_CONTROL
DEFINE SUB	Topic	MQZAO_CONTROL
DELETE SUB	Topic	MQZAO_CONTROL
DISPLAY SUB	Topic	MQZAO_DISPLAY

Security Commands

Command	Object	Authorization required
SET AUTHREC	Queue manager	MQZAO_CHANGE
DELETE AUTHREC	Queue manager	MQZAO_CHANGE
DISPLAY AUTHREC	Queue manager	MQZAO_DISPLAY
DISPLAY AUTHSERV	Queue manager	MQZAO_DISPLAY
DISPLAY ENTAUTH	Queue manager	MQZAO_DISPLAY
SET CHLAUTH	Queue manager	MQZAO_CHANGE
DISPLAY CHLAUTH	Queue manager	MQZAO_DISPLAY
REFRESH SECURITY	Queue manager	MQZAO_CHANGE

Status Displays

Command	Object	Authorization required
DISPLAY CHSTATUS	Queue manager	MQZAO_DISPLAY Note that `+inq` authority (or equivalently MQZAO_INQUIRE) is required on the transmission queue if the channel type is CLUSSDR.
DISPLAY LSSTATUS	Queue manager	MQZAO_DISPLAY
DISPLAY PUBSUB	Queue manager	MQZAO_DISPLAY

Command	Object	Authorization required
DISPLAY SBSTATUS	Queue manager	MQZAO_DISPLAY
DISPLAY SVSTATUS	Queue manager	MQZAO_DISPLAY
DISPLAY TPSTATUS	Queue manager	MQZAO_DISPLAY

Cluster Commands

Command	Object	Authorization required
DISPLAY CLUSQMGR	Queue manager	MQZAO_DISPLAY
REFRESH CLUSTER	'mqm' group membership required	
RESET CLUSTER	'mqm' group membership required	
SUSPEND QMGR	'mqm' group membership required	
RESUME QMGR	'mqm' group membership required	

Other Administrative Commands

Command	Object	Authorization required
PING QMGR	Queue manager	MQZAO_DISPLAY
REFRESH QMGR	Queue manager	MQZAO_CHANGE
RESET QMGR	Queue manager	MQZAO_CHANGE
DISPLAY CONN	Queue manager	MQZAO_DISPLAY
STOP CONN	Queue manager	MQZAO_CHANGE

Note:

1. For DEFINE commands, MQZAO_DISPLAY authority is also needed for the LIKE object if one is specified, or on the appropriate SYSTEM.DEFAULT.xxx object if LIKE is omitted.
2. The MQZAO_CREATE authority is not specific to a particular object or object type. Create authority is granted for all objects for a specified queue manager, by specifying an object type of QMGR on the `setmqaut` command.
3. This applies if the object to be replaced already exists. If it does not, the check is as for DEFINE *object* NOREPLACE.

Related information

Clustering: Using REFRESH CLUSTER best practices

ALW *Authorizations for PCF commands*

This section summarizes the authorizations needed for each PCF command.

No check means that no authorization checking is carried out; *Not applicable* means that this operation is not relevant to this object type.

The user ID under which the program that submits the command is running must also have the following authorities:

- MQZAO_CONNECT authority to the queue manager
- MQZAO_DISPLAY authority on the queue manager in order to perform PCF commands

The special authorization MQZAO_ALL_ADMIN includes all the authorizations in the following list that are relevant to the object type, except MQZAO_CREATE, which is not specific to a particular object or object type.

Change *object*

Object	Authorization required
Queue	MQZAO_CHANGE
Topic	MQZAO_CHANGE
Process	MQZAO_CHANGE
Queue manager	MQZAO_CHANGE
Namelist	MQZAO_CHANGE
Authentication information	MQZAO_CHANGE
Channel	MQZAO_CHANGE
Client connection channel	MQZAO_CHANGE
Listener	MQZAO_CHANGE
Service	MQZAO_CHANGE
Communication information	MQZAO_CHANGE

Clear *object*

Object	Authorization required
Queue	MQZAO_CLEAR
Topic	MQZAO_CLEAR
Process	Not applicable
Queue manager	Not applicable
Namelist	Not applicable
Authentication information	Not applicable
Channel	Not applicable
Client connection channel	Not applicable
Listener	Not applicable
Service	Not applicable
Communication information	Not applicable

Copy *object* (without replace) (1)

Object	Authorization required
Queue	MQZAO_CREATE **(2)**
Topic	MQZAO_CREATE **(2)**
Process	MQZAO_CREATE **(2)**
Queue manager	Not applicable
Namelist	MQZAO_CREATE **(2)**
Authentication information	MQZAO_CREATE **(2)**
Channel	MQZAO_CREATE **(2)**

Object	Authorization required
Client connection channel	MQZAO_CREATE (**2**)
Listener	MQZAO_CREATE (**2**)
Service	MQZAO_CREATE (**2**)
Communication information	MQZAO_CREATE (**“2” on page 144**)

Copy *object* (with replace) (1, 4)

Object	Authorization required
Queue	MQZAO_CHANGE
Topic	MQZAO_CHANGE
Process	MQZAO_CHANGE
Queue manager	Not applicable
Namelist	MQZAO_CHANGE
Authentication information	MQZAO_CHANGE
Channel	MQZAO_CHANGE
Client connection channel	MQZAO_CHANGE
Listener	MQZAO_CHANGE
Service	MQZAO_CHANGE
Communication information	MQZAO_CHANGE

Create *object* (without replace) (3)

Object	Authorization required
Queue	MQZAO_CREATE (**2**)
Topic	MQZAO_CREATE (**2**)
Process	MQZAO_CREATE (**2**)
Queue manager	Not applicable
Namelist	MQZAO_CREATE (**2**)
Authentication information	MQZAO_CREATE (**2**)
Channel	MQZAO_CREATE (**2**)
Client connection channel	MQZAO_CREATE (**2**)
Listener	MQZAO_CREATE (**2**)
Service	MQZAO_CREATE (**2**)
Communication information	MQZAO_CREATE (**2**)

Create *object* (with replace) (3, 4)

Object	Authorization required
Queue	MQZAO_CHANGE
Topic	MQZAO_CHANGE

Object	Authorization required
Process	MQZAO_CHANGE
Queue manager	Not applicable
Namelist	MQZAO_CHANGE
Authentication information	MQZAO_CHANGE
Channel	MQZAO_CHANGE
Client connection channel	MQZAO_CHANGE
Listener	MQZAO_CHANGE
Service	MQZAO_CHANGE
Communication information	MQZAO_CHANGE

Delete *object*

Object	Authorization required
Queue	MQZAO_DELETE
Topic	MQZAO_DELETE
Process	MQZAO_DELETE
Queue manager	Not applicable
Namelist	MQZAO_DELETE
Authentication information	MQZAO_DELETE
Channel	MQZAO_DELETE
Client connection channel	MQZAO_DELETE
Listener	MQZAO_DELETE
Service	MQZAO_DELETE
Communication information	MQZAO_DELETE

Inquire *object*

Object	Authorization required
Queue	MQZAO_DISPLAY
Topic	MQZAO_DISPLAY
Process	MQZAO_DISPLAY
Queue manager	MQZAO_DISPLAY
Namelist	MQZAO_DISPLAY
Authentication information	MQZAO_DISPLAY
Channel	MQZAO_DISPLAY
Client connection channel	MQZAO_DISPLAY
Listener	MQZAO_DISPLAY
Service	MQZAO_DISPLAY

Object	Authorization required
Communication information	MQZAO_DISPLAY

Inquire *object* names

Object	Authorization required
Queue	No check
Topic	No check
Process	No check
Queue manager	No check
Namelist	No check
Authentication information	No check
Channel	No check
Client connection channel	No check
Listener	No check
Service	No check
Communication information	No check

Start *object*

Object	Authorization required
Queue	Not applicable
Topic	Not applicable
Process	Not applicable
Queue manager	Not applicable
Namelist	Not applicable
Authentication information	Not applicable
Channel	MQZAO_CONTROL
Client connection channel	Not applicable
Listener	MQZAO_CONTROL
Service	MQZAO_CONTROL
Communication information	Not applicable

Stop *object*

Object	Authorization required
Queue	Not applicable
Topic	Not applicable
Process	Not applicable
Queue manager	Not applicable
Namelist	Not applicable

Object	Authorization required
Authentication information	Not applicable
Channel	MQZAO_CONTROL
Client connection channel	Not applicable
Listener	MQZAO_CONTROL
Service	MQZAO_CONTROL
Communication information	Not applicable

Channel Commands

Command	Object	Authorization required
Ping Channel	Channel	MQZAO_CONTROL
Reset Channel	Channel	MQZAO_CONTROL_EXTENDED
Resolve Channel	Channel	MQZAO_CONTROL_EXTENDED

Subscription Commands

Command	Object	Authorization required
Change Subscription	Topic	MQZAO_CONTROL
Create Subscription	Topic	MQZAO_CONTROL
Delete Subscription	Topic	MQZAO_CONTROL
Inquire Subscription	Topic	MQZAO_DISPLAY

Security Commands

Command	Object	Authorization required
Set Authority Record	Queue manager	MQZAO_CHANGE
Delete Authority Record	Queue manager	MQZAO_CHANGE
Inquire Authority Records	Queue manager	MQZAO_DISPLAY
Inquire Authority Service	Queue manager	MQZAO_DISPLAY
Inquire Entity Authority	Queue manager	MQZAO_DISPLAY
Set Channel Authentication Record	Queue manager	MQZAO_CHANGE
Inquire Channel Authentication Records	Queue manager	MQZAO_DISPLAY
Refresh Security	Queue manager	MQZAO_CHANGE

Status Displays

Command	Object	Authorization required
Inquire Channel Status	Queue manager	MQZAO_DISPLAY Note that +inq authority (or equivalently MQZAO_INQUIRE) is required on the transmission queue if the channel type is CLUSSDR.
Inquire Channel Listener Status	Queue manager	MQZAO_DISPLAY
Inquire Pub/Sub Status	Queue manager	MQZAO_DISPLAY
Inquire Subscription Status	Queue manager	MQZAO_DISPLAY
Inquire Service Status	Queue manager	MQZAO_DISPLAY
Inquire Topic Status	Queue manager	MQZAO_DISPLAY

Cluster Commands

Command	Object	Authorization required
Inquire Cluster Queue Manager	Queue manager	MQZAO_DISPLAY
Refresh Cluster	'mqm' group membership required	'mqm' group membership required
Reset Cluster	'mqm' group membership required	'mqm' group membership required
Suspend Queue Manager Cluster	'mqm' group membership required	'mqm' group membership required
Resume Queue Manager Cluster	'mqm' group membership required	'mqm' group membership required

Other Administrative Commands

Command	Object	Authorization required
Ping Queue Manager	Queue manager	MQZAO_DISPLAY
Refresh Queue Manager	Queue manager	MQZAO_CHANGE
Reset Queue Manager	Queue manager	MQZAO_CHANGE
Reset Queue Statistics	Queue	MQZAO_DISPLAY and MQZAO_CHANGE
Inquire Connection	Queue manager	MQZAO_DISPLAY
Stop Connection	Queue manager	MQZAO_CHANGE

Note:

1. For Copy commands, MQZAO_DISPLAY authority is also needed for the From object.
2. The MQZAO_CREATE authority is not specific to a particular object or object type. Create authority is granted for all objects for a specified queue manager, by specifying an object type of QMGR on the `setmqaut` command.
3. For Create commands, MQZAO_DISPLAY authority is also needed for the appropriate SYSTEM.DEFAULT.* object.

4. This applies if the object to be replaced already exists. If it does not, the check is as for Copy or Create without replace.

AIX Creating and managing groups on AIX

On AIX, providing you are not using NIS or NIS+, use SMITTY to work with groups.

About this task

On AIX, you can use SMITTY to create a group, add a user to a group, display a list of the users who are in the group, and remove a user from a group.

Procedure

1. From SMITTY, select **Security and Users** and press Enter.
2. Select **Groups** and press Enter.
3. To create a group, complete the following steps:
   a) Select **Add a Group** and press Enter.
   b) Enter the name of the group and the names of any users that you want to add to the group, separated by commas.
   c) Press Enter to create the group.
4. To add a user to a group, complete the following steps:
   a) Select **Change / Show Characteristics of Groups** and press Enter.
   b) Enter the name of the group to show a list of the members of the group.
   c) Add the names of the users that you want to add to the group, separated by commas.
   d) Press Enter to add the names to the group.
5. To display who is in a group, complete the following steps:
   a) Select **Change / Show Characteristics of Groups** and press Enter.
   b) Enter the name of the group to show a list of the members of the group.
6. To remove a user from a group, complete the following steps:
   a) Select **Change / Show Characteristics of Groups** and press Enter.
   b) Enter the name of the group to show a list of the members of the group.
   c) Delete the name of the user that you want to remove from the group.
   d) Press Enter to remove the name from the group.

Linux Creating and managing groups on Linux

On Linux, providing you are not using NIS or NIS+, use the `/etc/group` file to work with groups.

About this task

On Linux, group information is held in the `/etc/group` file. You can use commands to create a group, add a user to a group, display a list of the users who are in the group, and remove a user from a group.

Procedure

1. To create a new group, use the **groupadd** command.
   Type the following command:

   ```
   groupadd -g group-ID group-name
   ```

   where *group-ID* is the numeric identifier of the group, and *group-name* is the name of the group.

2. To add a member to a supplementary group, use the **usermod** command to list the supplementary groups that the user is currently a member of, and the supplementary groups that the user is to become a member of.
   For example, if the user is already a member of the group groupa, and is to become a member of groupb, use the following command:

   ```
   usermod -G groupa,groupb user-name
   ```

   where *user-name* is the user name.
3. To display who is a member of a group, use the **getent** command.

   Type the following command:

   ```
   getent group group-name
   ```

   where *group-name* is the name of the group.
4. To remove a member from a supplementary group, use the **usermod** command to list the supplementary groups that you want the user to remain a member of.
   For example, if the user's primary group is users and the user is also a member of the groups mqm, groupa and groupb, to remove the user from the mqm group, use the following command:

   ```
   usermod -G groupa,groupb user-name
   ```

   where *user-name* is the user name.

Windows Creating and managing groups on Windows

On Windows, you use the Computer Management feature to administer groups on a workstation or member server machine.

About this task

For domain controllers, users and groups are administered through Active Directory. For more details on using Active Directory refer to the appropriate operating system instructions.

Any changes you make to a principal's group membership are not recognized until the queue manager is restarted, or you issue the MQSC command **REFRESH SECURITY** (or the PCF equivalent).

Use the Windows Computer Management panel to work with user and groups. Any changes made to the current logged on user might not be effective until the user logs in again.

Windows *Creating a group on Windows*

Create a group by using the control panel.

Procedure

1. Open the control panel
2. Double-click **Administrative Tools**.

   The Administrative Tools panel opens.
3. Double-click **Computer Management**.

   The Computer Management panel opens.
4. Expand **Local Users and Groups**.
5. Right-click **Groups**, and select **New Group...**.

   The New Group panel is displayed.
6. Type an appropriate name in the Group name field, then click **Create**.
7. Click **Close**.

Windows *Adding a user to a group on Windows*

Add a user to a group by using the control panel.

Procedure

1. Open the control panel
2. Double-click **Administrative Tools**.

   The Administrative Tools panel opens.
3. Double-click **Computer Management**.

   The Computer Management panel opens.
4. From the Computer Management panel, expand **Local Users and Groups**.
5. Select **Users**
6. Double-click the user that you want to add to a group.

   The user properties panel is displayed.
7. Select the **Member Of** tab.
8. Select the group that you want to add the user to. If the group you want is not visible:

   a) Click **Add...**.

      The Select Groups panel is displayed.

   b) Click **Locations...**.

      The Locations panel is displayed.

   c) Select the location of the group you want to add the user to from the list and click **OK**.

   d) Type the group name in the field provided.

      Alternatively, click **Advanced...** and then **Find Now** to list the groups available in the currently selected location. From here, select the group you want to add the user to and click **OK**.

   e) Click **OK**.

      The user properties panel is displayed, showing the group you added.

   f) Select the group.
9. Click **OK**.

   The Computer Management panel is displayed.

Windows *Displaying who is in a group on Windows*

Display the members of a group by using the control panel.

Procedure

1. Open the control panel
2. Double-click **Administrative Tools**.

   The Administrative Tools panel opens.
3. Double-click **Computer Management**.

   The Computer Management panel opens.
4. From the Computer Management panel, expand **Local Users and Groups**.
5. Select **Groups**.
6. Double-click a group. The group properties panel is displayed.

   The group properties panel is displayed.

Results

The group members are displayed.

Removing a user from a group on Windows

Remove a user from a group by using the control panel.

Procedure

1. Open the control panel
2. Double-click **Administrative Tools**.

   The Administrative Tools panel opens.
3. Double-click **Computer Management**.

   The Computer Management panel opens.
4. From the Computer Management panel, expand **Local Users and Groups**.
5. Select **Users**.
6. Double-click the user that you want to add to a group.

   The user properties panel is displayed.
7. Select the **Member Of** tab.
8. Select the group that you want to remove the user from, then click **Remove**.
9. Click **OK**.

   The Computer Management panel is displayed.

Results

You have now removed the user from the group.

Special considerations for security on Windows

Some security functions behave differently on different versions of Windows.

IBM MQ security relies on calls to the operating system API for information about user authorizations and group memberships. Some functions do not behave identically on the Windows systems. This collection of topics includes descriptions of how those differences might affect IBM MQ security when you are running IBM MQ in a Windows environment.

Local and domain user accounts for the IBM MQ Windows service

When IBM MQ is running, it must check that only authorized users can access queue managers or queues. This requires a special user account that IBM MQ can use to query information about the any user attempting such access.

- “Configuring special user accounts with the Prepare IBM MQ Wizard” on page 148
- “Using IBM MQ with Active Directory” on page 149
- “User rights required for an IBM MQ Windows service” on page 149

Configuring special user accounts with the Prepare IBM MQ Wizard

The Prepare IBM MQ Wizard creates a special user account so that the Windows service can be shared by processes that need to use it (see Configuring IBM MQ with the PPrepare IBM MQ Wizard).

A Windows service is shared between client processes for an IBM MQ installation. One service is created for each installation. Each service is named `MQ_`*`InstallationName`*, and has a display name of `IBM MQ(`*`InstallationName`*`)`.

Because each service must be shared between non-interactive and interactive logon sessions, you must launch each under a special user account. You can use one special user account for all the services, or create different special user accounts. Each special user account must have the user right to `Logon as a service`, for more information see Table 14 on page 149. If the user ID does not have the authority to run the service, the service does not start and it returns an error in the Windows system event log. Typically, you will have run the Prepare IBM MQ Wizard, and set up the user ID correctly. However, if you have configured the user ID manually, is it possible that you might have a problem that you will need to resolve.

When you install IBM MQ and run the Prepare IBM MQ Wizard for the first time, it creates a local user account for the service called MUSR_MQADMIN with the required settings and permissions, including `Logon as a service`.

For subsequent installations, the Prepare IBM MQ Wizard creates a user account named MUSR_MQADMIN*x*, where *x* is the next available number representing a user ID that does not exist. The password for MUSR_MQADMIN*x* is randomly generated when the account is created, and used to configure the logon environment for the service. The generated password does not expire.

This IBM MQ account is not affected by any account policies that are set up on the system to require that account passwords are changed after a certain period.

The password is not known outside this one-time processing and is stored by the Windows operating system in a secure part of the registry.

Using IBM MQ with Active Directory

In some network configurations, where user accounts are defined on domain controllers that are using the Active Directory directory service, the local user account that IBM MQ is running under might not have the authority that it requires to query the group membership of other domain user accounts. When you install IBM MQ, the Prepare IBM MQ Wizard identifies whether this is the case by carrying out tests and asking you questions about the network configuration.

If the local user account that IBM MQ is running under does not have the required authority, the Prepare IBM MQ Wizard prompts you for the account details of a domain user account with particular user rights. For information about how to create and set up a Windows domain account, see Creating and setting up Windows domain accounts for IBM MQ. For the user rights that the domain user account requires, see Table 14 on page 149.

When you have entered valid account details for the domain user account into the Prepare IBM MQ Wizard, the wizard configures an IBM MQ Windows service to run under the new account. The account details are held in the secure part of the Registry and cannot be read by users.

When the service is running, an IBM MQ Windows service is launched and remains running for as long as the service is running. An IBM MQ administrator who logs on to the server after the Windows service is launched can use the IBM MQ Explorer to administer queue managers on the server. This connects the IBM MQ Explorer to the existing Windows service process. These two actions need different levels of permission before they can work:

- The launch process requires a launch permission.
- The IBM MQ administrator requires Access permission.

User rights required for an IBM MQ Windows service

The following table lists the user rights required for the local and domain user accounts under which the Windows service for an IBM MQ installation runs.

Table 14. User rights required for an IBM MQ Windows service

Permission	**Description**
Log on as batch job	Enables an IBM MQ Windows service to run under this user account.
Log on as service	Enables users to set the IBM MQ Windows service to log on using the configured account.
Shut down the system	Allows the IBM MQ Windows service to restart the server if configured to do so when recovery of a service fails.

Table 14. User rights required for an IBM MQ Windows service (continued)

Permission	Description
Increase quotas	Required for operating system `CreateProcessAsUser` call.
Act as part of the operating system	Required for operating system `LogonUser` call.
Bypass traverse checking	Required for operating system `LogonUser` call.
Replace a process level token	Required for operating system `LogonUser` call.

Note: Debug programs rights might be needed in environments running ASP and IIS applications.

Your domain user account must have these Windows user rights set as effective user rights as listed in the Local Security Policy application. If they are not, set them using either the Local Security Policy application locally on the server, or by using the Domain Security Application domain wide.

Windows *Windows Server security permissions*
Installation of IBM MQ behaves differently on Windows Server, depending on whether a local user or domain user performs the installation.

If a *local* user installs IBM MQ, the Prepare IBM MQ Wizard detects that the local user created for the IBM MQ Windows service can retrieve the group membership information of the installing user. The Prepare IBM MQ Wizard asks the user questions about the network configuration to determine whether there are other user accounts defined on domain controllers running on Windows 2000 or later. If so, the IBM MQ Windows service needs to run under a domain user account with particular settings and authorities. The Prepare IBM MQ Wizard prompts the user for the account details of this user as described in Configuring IBM MQ with the Prepare IBM MQ Wizard.

If a *domain* user installs IBM MQ, the Prepare IBM MQ Wizard detects that the local user created for the IBM MQ Windows service cannot retrieve the group membership information of the installing user. In this case, the Prepare IBM MQ Wizard always prompts the user for the account details of the domain user account for the IBM MQ Windows service to use.

When the IBM MQ Windows service needs to use a domain user account, IBM MQ cannot operate correctly until this has been configured using the Prepare IBM MQ Wizard. The Prepare IBM MQ Wizard does not allow the user to continue with other tasks, until the Windows service has been configured with a suitable account.

For more information, see Creating and setting up domain accounts for IBM MQ.

Windows *Changing the user name associated with the IBM MQ service*
You can change the user name associated with the IBM MQ service by creating a new account and entering its details using the Prepare IBM MQ Wizard.

About this task

When you install IBM MQ and run the Prepare IBM MQ Wizard for the first time, it creates a local user account for the service called MUSR_MQADMIN. For subsequent installations, the Prepare IBM MQ Wizard creates a user account named MUSR_MQADMIN*x*, where *x* is the next available number representing a user ID that does not exist.

You might need to change the user name associated with the IBM MQ service from MUSR_MQADMIN or MUSR_MQADMIN*x* to something else. For example, you might need to do this if your queue manager is associated with Db2®, which does not accept user names of more than 8 characters.

Procedure

1. Create a new user account (for example **NEW_NAME**)

2. Use the Prepare IBM MQ Wizard to enter the details of the new user account.

Related tasks

Configuring IBM MQ with the Prepare IBM MQ Wizard

Windows *Changing the password of the IBM MQ Windows service local user account*
You can change the password of the IBM MQ Windows service local user account by using the Computer Management panel.

About this task

To change the password of the IBM MQ Windows service local user account, perform the following steps:

Procedure

1. Identify the user the service is running under.
2. Stop the IBM MQ service from the Computer Management panel.
3. Change the required password in the same way that you would change the password of an individual.
4. Go to the properties for the IBM MQ service from the Computer Management panel.
5. Select the **Log On** page.
6. Confirm that the account name specified matches the user for which the password was modified.
7. Type the password into the **Password** and **Confirm password** fields and click **OK**.

Windows *Changing the password for an IBM MQ Windows service for an installation running under a domain user account*
As an alternative to using the Prepare IBM MQ Wizard to enter the account details of the domain user account, you can use the Computer Management panel to alter the **Log On** details for the installation specific IBM MQ Service.

About this task

If the IBM MQ Windows service for an installation is running under a domain user account, you can change the password for the account as follows:

Procedure

1. Change the password for the domain account on the domain controller. You might need to ask your domain administrator to do this for you.
2. Complete the following the steps to modify the **Log On** page for the IBM MQ service.
   a) Identify the user that the service is running under.
   b) Stop the IBM MQ service from the Computer Management panel.
   c) Change the required password in the same way that you would change the password of an individual.
   d) Go to the properties for the IBM MQ service from the Computer Management panel.
   e) Select the **Log On** page.
   f) Confirm that the account name specified matches the user for which the password was modified.
   g) Type the password into the **Password** and **Confirm password** fields and click **OK**.

   The user account that IBM MQ Windows service runs under executes any MQSC commands that are issued by user interface applications, or performed automatically on system startup, shutdown, or service recovery. This user account must therefore have IBM MQ administration rights. By default it is added to the local mqm group on the server. If this membership is removed, the IBM MQ Windows service does not work. For more information about user rights, see "User rights required for an IBM MQ Windows service" on page 149.

If a security problem arises with the user account that the IBM MQ Windows service runs under, error messages and descriptions appear in the system event log.

Related tasks

Configuring IBM MQ with the Prepare IBM MQ Wizard

Windows *Considerations when promoting Windows servers to domain controllers*

When promoting a Windows server to a domain controller, you should consider whether the security setting relating to user and group permissions is appropriate. When changing the state of a Windows machine between server and domain controller, you should take into consideration that this can affect the operation of IBM MQ because IBM MQ uses a locally-defined mqm group.

Security settings relating to domain user and group permissions

IBM MQ relies on group membership information to implement its security policy, which means that it is important that the user ID that is performing IBM MQ operations can determine the group memberships of other users.

When you promote a Windows server to a domain controller, you are presented with an option for the security setting relating to user and group permissions. This option controls whether arbitrary users are able to retrieve group memberships from the active directory. If a domain controller is set up so that local accounts do have the authority to query the group membership of the domain user accounts, the default user ID created by IBM MQ during the installation process can obtain group memberships for other users as required. However, if a domain controller is set up so that local accounts do not have the authority to query the group membership of the domain user accounts, this prevents IBM MQ from completing its checks that users who are defined on the domain are authorized to access queue managers or queues, and access fails. If you are using Windows on a domain controller that has been set up in this way, a special domain user account with the required permissions must be used.

In this case, you need to know:

- How security permissions for your version of Windows behave.
- How to allow domain mqm group members to read group membership.
- How to configure an IBM MQ Windows service to run under a domain user.

For more information, see Configuring user accounts for IBM MQ.

IBM MQ access to the local mqm group

When Windows servers are promoted to, or demoted from, domain controllers, IBM MQ loses access to the local mqm group.

When a server is promoted to be a domain controller, the scope changes from local to domain local. When the machine is demoted to server, all domain local groups are removed. This means that changing a machine from server to domain controller and back to server loses access to a local mqm group. The symptom is an error indicating the lack of a local mqm group, for example:

```
>crtmqm qm0
AMQ8066:Local mqm group not found.
```

To remedy this problem, re-create the local mqm group using the standard Windows management tools. Because all group membership information is lost, you must reinstate privileged IBM MQ users in the newly-created local mqm group. If the machine is a domain member, you must also add the domain mqm group to the local mqm group to grant privileged domain IBM MQ user IDs the required level of authority.

Windows *Restrictions on nested groups on Windows*

There are restrictions on the use of nested groups. These result partly from the domain functional level and partly from IBM MQ restrictions.

Active Directory can support different group types within a Domain context depending on the Domain functional level. By default, Windows 2003 domains are in the " Windows 2000 mixed" functional level. (Windows Server 2008 and Windows Server 2012 follow the Windows 2003 domain model.) The domain functional level determines the supported group types and level of nesting allowed when configuring user IDs in a domain environment. Refer to Active Directory documentation for details on the Group Scope and inclusion criteria.

In addition to Active Directory requirements, further restrictions are imposed on IDs used by IBM MQ. The network APIs used by IBM MQ do not support all the configurations that are supported by the domain functional level. As a result, IBM MQ is not able to query the group memberships of any Domain IDs present in a Domain Local group which is then nested in a local group. Furthermore, multiple nesting of global and universal groups is not supported. However, immediately nested global or universal groups are supported.

Windows *Authorizing users to use IBM MQ remotely*

If you need to create and start queue managers when connected to IBM MQ remotely, you must have the `Create global objects` user access.

About this task

Note: Administrators have the `Create global objects` user access by default, so if you are an administrator you can create and start queue managers when connected remotely without altering your user rights.

If you are connecting to a Windows machine using either Terminal Services or a Remote Desktop Connection and you have problems creating, starting or deleting a queue manager this might be because you do not have the user access `Create global objects`.

The `Create global objects` user access limits the users authorized to create objects in the global namespace. In order for an application to create a global object, it must either be running in the global namespace, or the user under which the application is running must have the `Create global objects` user access applied to it.

When you connect remotely to a Windows machine using either Terminal Services or Remote Desktop Connection, applications run in their own local namespace. If you attempt to create or delete a queue manager using IBM MQ Explorer or the **`crtmqm`** or **`dltmqm`** command, or to start a queue manager using the **`strmqm`** command, it results in an authorization failure. This creates an IBM MQ FDC with Probe ID XY132002.

Starting a queue manager using the IBM MQ Explorer, or using the **`amqmdain qmgr start`** command works correctly because these commands do not directly start the queue manager. Instead the commands send the request to start the queue manager to a separate process running in the global namespace.

If the various methods of administering IBM MQ do not work when you use terminal services, try setting the `Create global objects` user right.

Procedure

1. Open the Administrative Tools panel:

   Windows Server 2008 and Windows Server 2012
   Access this panel using **Control Panel** > **System and Maintenance** > **Administrative Tools**.

   Windows 8.1
   Access this panel using **Administrative Tools** > **Computer Management**
2. Double-click **Local Security Policy**.

3. Expand **Local Policies**.
4. Click **User Rights Assignment**.
5. Add the new user or group to the `Create global objects` policy.

Windows *The SSPI channel exit program on Windows*

IBM MQ for Windows supplies a security exit program, which can be used on both message and MQI channels. The exit is supplied as source and object code, and provides one-way and two-way authentication.

The security exit uses the Security Support Provider Interface (SSPI), which provides the integrated security facilities of Windows platforms.

The security exit provides the following identification and authentication services:

One way authentication

This uses Windows NT LAN Manager (NTLM) authentication support. NTLM allows servers to authenticate their clients. It does not allow a client to authenticate a server, or one server to authenticate another. NTLM was designed for a network environment in which servers are assumed to be genuine. NTLM is supported on all Windows platforms that are supported by IBM WebSphere MQ 7.0.

This service is typically used on an MQI channel to enable a server queue manager to authenticate an IBM MQ MQI client application. A client application is identified by the user ID associated with the process that is running.

To perform the authentication, the security exit at the client end of a channel acquires an authentication token from NTLM and sends the token in a security message to its partner at the other end of the channel. The partner security exit passes the token to NTLM, which checks that the token is authentic. If the partner security exit is not satisfied with the authenticity of the token, it instructs the MCA to close the channel.

Two way, or mutual, authentication

This uses Kerberos authentication services. The Kerberos protocol does not assume that servers in a network environment are genuine. Servers can authenticate clients and other servers, and clients can authenticate servers. Kerberos is supported on all Windows platforms that are supported by IBM WebSphere MQ 7.0.

This service can be used on both message and MQI channels. On a message channel, it provides mutual authentication of the two queue managers. On an MQI channel, it enables the server queue manager and the IBM MQ MQI client application to authenticate each other. A queue manager is identified by its name prefixed by the string `ibmMQSeries/`. A client application is identified by the user ID associated with the process that is running.

To perform the mutual authentication, the initiating security exit acquires an authentication token from the Kerberos security server and sends the token in a security message to its partner. The partner security exit passes the token to the Kerberos server, which checks that it is authentic. The Kerberos security server generates a second token, which the partner sends in a security message to the initiating security exit. The initiating security exit then asks the Kerberos server to check that the second token is authentic. During this exchange, if either security exit is not satisfied with the authenticity of the token sent by the other, it instructs the MCA to close the channel.

The security exit is supplied in both source and object format. You can use the source code as a starting point for writing your own channel exit programs or you can use the object module as supplied. The object module has two entry points, one for one way authentication using NTLM authentication support and the other for two way authentication using Kerberos authentication services.

For more information about how the SSPI channel exit program works, and for instructions on how to implement it, see Using the SSPI security exit on Windows systems.

Windows *Applying security template files on Windows*

Applying a template might affect the security settings applied to IBM MQ files and directories. If you use the highly secure template, apply it before installing IBM MQ.

Windows supports text-based security template files that you can use to apply uniform security settings to one or more computers with the Security Configuration and Analysis MMC snap-in. In particular, Windows supplies several templates that include a range of security settings with the aim of providing specific levels of security. These templates include Compatible, Secure, and Highly Secure.

Applying one of these templates might affect the security settings applied to IBM MQ files and directories. If you want to use the Highly Secure template, configure your machine before you install IBM MQ.

If you apply the highly secure template to a machine on which IBM MQ is already installed, all the permissions you have set on the IBM MQ files and directories are removed. Because these permissions are removed, you lose *Administrator*, *mqm*, and, when applicable, *Everyone* group access from the error directories.

Windows *Configuring extra authority for Windows applications connecting to IBM MQ*

The account under which IBM MQ processes run might need extra authorization before SYNCHRONIZE access to application processes can be granted.

About this task

You might experience problems if you have Windows applications, for example ASP pages, connecting to IBM MQ that are configured to run at a security level higher than usual.

IBM MQ requires SYNCHRONIZE access to application processes in order to coordinate certain actions. When a server application first attempts to connect to a queue manager IBM MQ modifies the process to grant SYNCHRONIZE authority for IBM MQ administrators. However, the account under which IBM MQ processes run might need additional authorization before the requested access can be granted.

To configure additional authority to the user ID under which IBM MQ processes are running, complete the following steps:

Procedure

1. Start the Local Security Policy tool, click **Security Settings**->**Local Policies**->**User Right Assignments**, the click **Debug Programs**.
2. Double-click **Debug Programs**, then add your IBM MQ user ID to the list

   If the system is in a Windows domain and the effective policy setting is still not set, even though the local policy setting is set, the user ID must be authorized in the same way at domain level, using the Domain Security Policy tool.

IBM i Setting up security on IBM i

Security on IBM i is implemented using the IBM MQ Object Authority Manager (OAM) and IBM i object level security.

Security considerations that must be made when determining access authority to IBM MQ objects.

You need to consider the following points when setting up authorities to the users in your enterprise:

1. Grant and revoke authorities to the IBM MQ for IBM i commands using the IBM i `GRTOBJAUT` and `RVKOBJAUT` commands.

   In the `QMQM` library, certain noncommand (*cmd) objects are set to have **`*PUBLIC`** authority to **`*USE`**. Do not change the authorities of these objects or use an authorization list to provide authority. Any incorrect authority might compromise IBM MQ functionality.
2. During installation of IBM MQ for IBM i, the following special user profiles are created:

QMQM
Is used primarily for internal product-only functions. However, it can be used to run trusted applications using MQCNO_FASTPATH_BINDINGS. See Connecting to a queue manager using the MQCONNX call.

QMQMADM
Is used as a group profile for administrators of IBM MQ. The group profile gives access to CL commands and IBM MQ resources.

When using SBMJOB to submit programs that call IBM MQ commands, USER must not be set explicitly to QMQMADM. Instead, set USER to QMQM or another user profile that has QMQMADM specified as a group.

3. If you are sending channel commands to remote queue managers, ensure that your user profile is a member of the group QMQMADM on the target system. For a list of PCF and MQSC channel commands, see IBM MQ for IBM i CL commands.
4. The group set associated with a user is cached when the group authorizations are computed by the OAM.

   Any changes made to a user's group memberships after the group set has been cached are not recognized until you restart the queue manager or execute RFRMQMAUT to refresh security.
5. Limit the number of users who have authority to work with commands that are particularly sensitive. These commands include:
   - Create Message Queue Manager (`CRTMQM`)
   - Delete Message Queue Manager (`DLTMQM`)
   - Start Message Queue Manager (`STRMQM`)
   - End Message Queue Manager (`ENDMQM`)
   - Start Command Server (`STRMQMCSVR`)
   - End Command Server (`ENDMQMCSVR`)
6. Channel definitions contain a security exit program specification. Channel creation and modification requires special considerations. Details of security exits are given in "Security exit overview" on page 110.
7. The channel exit and trigger monitor programs can be substituted. The security of such replacements is the responsibility of the programmer.

IBM i Object authority manager on IBM i

The object authority manager (OAM) manages users' authorizations to manipulate IBM MQ objects, including queues and process definitions. It also provides a command interface through which you can grant or revoke access authority to an object for a specific group of users. The decision to allow access to a resource is made by the OAM, and the queue manager follows that decision. If the OAM cannot make a decision, the queue manager prevents access to that resource.

Through the OAM you can control:

- Access to IBM MQ objects through the MQI. When an application program attempts to access an object, the OAM checks that the user profile making the request has the authorization for the operation requested.

  In particular, this means that queues, and the messages on queues, can be protected from unauthorized access.
- Permission to use PCF and MQSC commands.

Different groups of users can have different access authority to the same object. For example, for a specific queue, one group could perform both put and get operations; another group might be allowed only to browse the queue (MQGET with browse option). Similarly, some groups might have get and put authority to a queue, but not be allowed to alter or delete the queue.

IBM MQ for IBM i commands and perform operations on IBM MQ for IBM i objects

IBM MQ authorities on IBM i

To access IBM MQ objects, you need authority to issue the command and to access the object referenced. Administrators have access to all IBM MQ resources.

Access to IBM MQ objects is controlled by authorities to:

1. Issue the IBM MQ command
2. Access the IBM MQ objects referenced by the command

All IBM MQ for IBM i CL commands are shipped with an owner of QMQM, and the administration profile (QMQMADM) has *USE rights with the *PUBLIC access set to *EXCLUDE.

Note: The QSRDUPER program is used by the IBM MQ for IBM i licensed program installer to duplicate Command (*CMD) objects in QSYS. In IBM i V5R4 and later, the QSRDUPER program was changed so that the default behavior is to create a proxy command rather than a duplicate of the original command. A proxy command redirects command execution to another command and has an attribute of PRX.
If a proxy command by the same name as the command being copied exists in library QSYS, private authorities to the proxy command are not granted to the command in the product library. Attempts to prompt or run the proxy command in QSYS check the authority of the target command in the product library. Any changes in authority to *CMD objects therefore need to be done in the product library (QMQM) and those in QSYS do not need to be modified. For example:

```
GRTOBJAUT OBJ(QMQM/DSPMQMQ) OBJTYPE(*CMD) USER(MQUSER) AUT(*USE)
```

Changes to the authority structure of some of the product's CL commands allows public use of these commands, if you have the required OAM authority to the IBM MQ objects to make these changes.

To be an IBM MQ administrator on IBM i, you must be a member of the *QMQMADM group*. This group has properties like the properties of the mqm group on AIX, Linux, and Windows systems. In particular, the QMQMADM group is created when you install IBM MQ for IBM i, and members of the QMQMADM group have access to all IBM MQ resources on the system. You also have access to all IBM MQ resources if you have *ALLOBJ authority.

Administrators can use CL commands to administer IBM MQ. One of these commands is GRTMQMAUT, which is used to grant authorities to other users. Another command, STRMQMMQSC, enables an administrator to issue MQSC commands to a local queue manager.

Related concepts

"Authority to administer IBM MQ on IBM i" on page 90

Access authorities for IBM MQ objects on IBM i

Access authorities required for running IBM MQ CL commands.

IBM MQ for IBM i categorizes the product's CL commands into two groups:

Group 1

Users must be in the QMQMADM user group, or have *ALLOBJ authority, to process these commands. Users having either of these authorities can process all commands in all categories without requiring any extra authority.

Note: These authorities override any OAM authority.

These commands can be grouped as follows:

- Command Server Commands
  - ENDMQMCSVR, End IBM MQ Command Server
  - STRMQMCSVR, Start IBM MQ Command Server
- Dead-Letter Queue Handler Command
  - STRMQMDLQ, Start IBM MQ Dead-Letter Queue Handler

- Listener Command
  - ENDMQMLSR, End IBM MQ listener
  - STRMQMLSR, Start non-object listener
- Media Recovery Commands
  - RCDMQMIMG, Record IBM MQ Object Image
  - RCRMQMOBJ, Re-create IBM MQ Object
  - WRKMQMTRN, Work with IBM MQ Q Transactions
- Queue Manager Commands
  - CRTMQM, Create Message Queue Manager
  - DLTMQM, Delete Message Queue Manager
  - ENDMQM, End Message Queue Manager
  - STRMQM, Start Message Queue Manager
- Security Commands
  - GRTMQMAUT, Grant IBM MQ Object Authority
  - RVKMQMAUT, Revoke IBM MQ Object Authority
- Trace Command
  - TRCMQM, Trace IBM MQ Job
- Transaction Commands
  - RSVMQMTRN, Resolve IBM MQ Transaction
- Trigger Monitor Commands
  - STRMQMTRM, Start Trigger Monitor
- IBM MQSC Commands
  - RUNMQSC, Run IBM MQSC Commands
  - STRMQMMQSC, Start IBM MQSC Commands

Group 2

The rest of the commands, for which two levels of authority are required:

1. IBM i authority to run the command. An IBM MQ administrator sets this using the **GRTOBJAUT** command to override the *PUBLIC(*EXCLUDE) restriction for a user or group of users.

   For example:

   ```
   GRTOBJAUT OBJ(QMQM/DSPMQMQ) OBJTYPE(*CMD) USER(MQUSER) AUT(*USE)
   ```

2. IBM MQ authority to manipulate the IBM MQ objects associated with the command, or commands, given the correct IBM i authority in Step 1.

   This authority is controlled by the user having the appropriate OAM authority for the required action, set by an IBM MQ administrator using the **GRTMQMAUT** command

   For example:

   ```
   GRTMQMAUT *connect authority to the queue manager + *admchg authority to
           the queue
   ```

The commands can be grouped as follows:

- Channel Commands
  - CHGMQMCHL, Change IBM MQ Channel

This requires *connect authority to the queue manager and *admchg authority to the channel.

– CPYMQMCHL, Copy IBM MQ Channel

  This requires *connect and *admcrt authority to the queue manager, *admdsp authority to the default channel type to be copied, and *admcrt authority to the channel object class.

  For example, copying a Sender channel, needs *admdsp authority to SYSTEM.DEF.SENDER channel

– CRTMQMCHL, Create IBM MQ Channel

  This requires *connect and *admcrt authority to the queue manager, *admdsp authority to the default channel type to be created and *admcrt authority to the channel object class.

  For example, creating a Sender channel, needs *admdsp authority to SYSTEM.DEF.SENDER channel

– DLTMQMCHL, Delete IBM MQ Channel

  This requires *connect authority to the queue manager and *admdlt authority to the channel.

– RSVMQMCHL, Resolve IBM MQ Channel

  This requires *connect authority to the queue manager and *ctrlx authority to the channel.

• Display commands

  To process the DSP commands you must grant the user `*connect` and `*admdsp` authority to the queue manager, together with any specific option listed:

  – DSPMQM, Display Message Queue Manager
  – DSPMQMAUT, Display IBM MQ Object Authority
  – DSPMQMAUTI, Display IBM MQ Authentication Information - `*admdsp` to the authentication information object
  – DSPMQMCHL, Display IBM MQ Channel - `*admdsp` to the channel
  – DSPMQMCSVR, Display IBM MQ Command Server
  – DSPMQMNL, Display IBM MQ Namelist - `*admdsp` to the namelist
  – DSPMQMOBJN, Display IBM MQ Object Names
  – DSPMQMPRC, Display IBM MQ Process - `*admdsp` to the process
  – DSPMQMQ, Display IBM MQ Queue - `*admdsp` to the queue
  – DSPMQMTOP, Display IBM MQ Topic - `*admdsp` to the topic

• Work with commands

  To process the WRK commands and display the options panel you must grant the user `*connect` and `*admdsp` authority to the queue manager, together with any specific option listed:

  – WRKMQM, Work with Message Queue Managers
  – WRKMQMAUT, Work with IBM MQ Object Authority
  – WRKMQMAUTD, Work with IBM MQ Object Authority Data
  – WRKMQMAUTI, Work with IBM MQ Authentication Information
    - `*admchg` for the Change IBM MQ Authentication Information Object command.
    - `*admcrt` for the Create and Copy IBM MQ Authentication Information Object command.
    - `*admdlt` for the Delete IBM MQ Authentication Information Object command.
    - `*admdsp` for the Display IBM MQ Authentication Information Object command.
  – WRKMQMCHL, Work with IBM MQ Channel

    This requires the following authorities:

    - `*admchg` for the Change IBM MQ Channel command.

  - *admclr for the Clear IBM MQ Channel command.
  - *admcrt for the Create and Copy IBM MQ Channel command.
  - *admdlt for the Delete IBM MQ Channel command.
  - *admdsp for the Display IBM MQ Channel command.
  - *ctrl for the Start IBM MQ Channel command.
  - *ctrl for the End IBM MQ Channel command.
  - *ctrl for the Ping IBM MQ Channel command.
  - *ctrlx for the Reset IBM MQ Channel command.
  - *ctrlx for the Resolve IBM MQ Channel command.
- WRKMQMCHST, Work with IBM MQ Channel Status

  This requires *admdsp authority to the channel.
- WRKMQMCL, Work with IBM MQ Clusters
- WRKMQMCLQ, Work with IBM MQ Cluster Queues
- WRKMQMCLQM, Work with IBM MQ Cluster Queue Manager
- WRKMQMLSR, Work with IBM MQ Listener
- WRKMQMMSG, Work with IBM MQ Messages

  This requires *browse authority to the queue
- WRKMQMNL, Work with IBM MQ Namelists

  This requires the following authorities:

  - *admchg for the Change IBM MQ Namelist command.
  - *admcrt for the Create and Copy IBM MQ Namelist command.
  - *admdlt for the Delete IBM MQ Namelist command.
  - *admdsp for the Display IBM MQ Namelist command.
- WRKMQMPRC, Work with IBM MQ Processes

  This requires the following authorities:

  - *admchg for the Change IBM MQ Process command.
  - *admcrt for the Create and Copy IBM MQ Process command.
  - *admdlt for the Delete IBM MQ Process command.
  - *admdsp for the Display IBM MQ Process command.
- WRKMQMQ, Work with IBM MQ queues

  This requires the following authorities:

  - *admchg for the Change IBM MQ Queue command.
  - *admclr for the Clear IBM MQ Queue command.
  - *admcrt for the Create and Copy IBM MQ Queue command.
  - *admdlt for the Delete IBM MQ Queue command.
  - *admdsp for the Display IBM MQ Queue command.
- WRKMQMQSTS, Work with IBM MQ Queue Status
- WRKMQMTOP, Work with IBM MQ Topics

  This requires the following authorities

  - *admchg for the Change IBM MQ Topic command.
  - *admcrt for the Create and Copy IBM MQ Topic command.
  - *admdlt for the Delete IBM MQ Topic command.

    - `*admdsp` for the Display IBM MQ Topic command.
  - WRKMQMSUB, Work with IBM MQ Subscriptions
- Other Channel commands

  To process the channel commands you must grant the user the specific authorities listed:

  - ENDMQMCHL, End IBM MQ Channel

    This requires `*connect` authority to the queue manager and `*allmqi` authority to the transmission queue associated with the channel.
  - ENDMQMLSR, End IBM MQ Listener

    This requires `*connect` authority to the queue manager and `*ctrl` authority to the named listener object.
  - PNGMQMCHL, Ping IBM MQ Channel

    This requires `*connect` and `*inq` authority to the queue manager and `*ctrl` authority to the channel object.
  - RSTMQMCHL, Reset IBM MQ Channel

    This requires `*connect` authority to the queue manager.
  - STRMQMCHL, Start IBM MQ Channel

    This requires `*connect` authority to the queue manager and `*ctrl` authority to the channel object.
  - STRMQMCHLI, Start IBM MQ Channel Initiator

    This requires `*connect` and `*inq` authority to the queue manager, and `*allmqi` authority to the initiation queue associated with the transmission queue of the channel.
  - STRMQMLSR, Start IBM MQ Listener

    This requires *connect authority to the queue manager and *ctrl authority to the named listener object.
- Other commands:

  To process the following commands you must grant the user the specific authorities listed:

  - CCTMQM, Connect to Message Queue Manager

    This requires no IBM MQ object authority.
  - CHGMQM, Change Message Queue Manager

    This requires `*connect` and `*admchg` authority to the queue manager.
  - CHGMQMAUTI, Change IBM MQ Authentication Information

    This requires `*connect` authority to the queue manager and `*admchg` and `*admdsp` authority to the authentication information object.
  - CHGMQMNL, Change IBM MQ Namelist

    This requires `*connect` authority to the queue manager and `*admchg` authority to the namelist.
  - CHGMQMPRC, Change IBM MQ Process

    This requires `*connect` authority to the queue manager and `*admchg` authority to the process.
  - CHGMQMQ, Change IBM MQ Queue

    This requires `*connect` authority to the queue manager and `*admchg` authority to the queue.
  - CLRMQMQ, Clear IBM MQ Queue

    This requires `*connect` authority to the queue manager and `*admclr` authority to the queue.
  - CPYMQMAUTI, Copy IBM MQ Authentication Information

This requires *connect authority to the queue manager and *admdsp authority to the authentication information object and *admcrt authority to the authentication information object class.

- CPYMQMNL, Copy IBM MQ Namelist

  This requires *connect and *admcrt authority to the queue manager.
- CPYMQMPRC, Copy IBM MQ Process

  This requires *connect and *admcrt authority to the queue manager.
- CPYMQMQ, Copy IBM MQ Queue

  This requires *connect and *admcrt authority to the queue manager.
- CRTMQMAUTI, Create IBM MQ Authentication Information

  This requires *connect authority to the queue manager and *admdsp authority to the authentication information object and *admcrt authority to the authentication information object class.
- CRTMQMNL, Create IBM MQ Namelist

  This requires *connect and *admcrt authority to the queue manager and *admdsp authority to the default namelist.
- CRTMQMPRC, Create IBM MQ Process

  This requires *connect and *admcrt authority to the queue manager and *admdsp authority to the default process.
- CRTMQMQ, Create IBM MQ Queue

  This requires *connect and *admcrt authority to the queue manager and *admdsp authority to the default queue.
- CVTMQMDTA, Convert IBM MQ Data Type Command

  This requires no IBM MQ object authority.
- DLTMQMAUTI, Delete IBM MQ Authentication Information

  This requires *connect authority to the queue manager and *ctrlx authority to the authentication information object.
- DLTMQMNL, Delete IBM MQ Namelist

  This requires *connect authority to the queue manager and *admdlt authority to the namelist.
- DLTMQMPRC, Delete IBM MQ Process

  This requires *connect authority to the queue manager and *admdlt authority to the process.
- DLTMQMQ, Delete IBM MQ Queue

  This requires *connect authority to the queue manager and *admdlt authority to the queue.
- DSCMQM, Disconnect from Message Queue Manager

  This requires no IBM MQ object authority.
- RFRMQMAUT, Refresh Security

  This requires *connect authority to the queue manager.
- RFRMQMCL, Refresh Cluster

  This requires *connect authority to the queue manager.
- RSMMQMCLQM, Resume Cluster Queue Manager

  This requires *connect authority to the queue manager.
- RSTMQMCL, Reset Cluster

  This requires *connect authority to the queue manager.

– SPDMQMCLQM, Suspend Cluster Queue Manager

  This requires *connect authority to the queue manager.

IBM i Access authorizations on IBM i

Use this information to understand the access authorization commands.

Authorizations defined by the AUT keyword on the GRTMQMAUT and RVKMQMAUT commands can be categorized as follows:

- Authorizations related to MQI calls
- Authorization-related administration commands
- Context authorizations
- General authorizations, that is, for MQI calls, for commands, or both

The following tables list the different authorities, using the AUT parameter for MQI calls, Context calls, MQSC and PCF commands, and generic operations.

Table 15. Authorizations for MQI calls

AUT	Description
*ALTUSR	Allow another user's authority to be used for MQOPEN and MQPUT1 calls.
*BROWSE	Retrieve a message from a queue by issuing an MQGET call with the BROWSE option.
*CONNECT	Connect the application to the specified queue manager by issuing an MQCONN call.
*GET	Retrieve a message from a queue by issuing an MQGET call.
*INQ	Make an inquiry on a specific queue by issuing an MQINQ call.
*PUB	Open a topic to publish a message using an MQPUT call.
*PUT	Put a message on a specific queue by issuing an MQPUT call.
*RESUME	Resume a subscription using an MQSUB call.
*SET	Set attributes on a queue from the MQI by issuing an MQSET call. If you open a queue for multiple options, you must be authorized for each of them.
*SUB	Create, Alter or Resume a subscription to a topic using an MQSUB call.

Table 16. Authorizations for context calls

AUT	Description
*PASSALL	Pass all context on the specified queue. All the context fields are copied from the original request.
*PASSID	Pass identity context on the specified queue. The identity context is the same as that of the request.
*SETALL	Set all context on the specified queue. This is used by special system utilities.
*SETID	Set identity context on the specified queue. This is used by special system utilities.

Table 17. Authorizations for MQSC and PCF calls

AUT	Description
*ADMCHG	Change the attributes of the specified object.

Table 17. Authorizations for MQSC and PCF calls (continued)

AUT	Description
*ADMCLR	Clear the specified object (PCF Clear object command only).
*ADMCRT	Create objects of the specified type.
*ADMDLT	Delete the specified object.
*ADMDSP	Display the attributes of the specified object.

Table 18. Authorizations for generic operations

AUT	Description
*ALL	Use all operations applicable to the object. `all` authority is equivalent to the union of the authorities `alladm`, `allmqi`, and `system` appropriate to the object type.
*ALLADM	Perform all administration operations applicable to the object.
*ALLMQI	Use all MQI calls applicable to the object.
*CTRL	Control startup and shutdown of channels, listeners, and services.
*CTRLX	Reset sequence number and resolve indoubt channels.

IBM i *Using the access authorization commands on IBM i*

Use this information to learn about the access authorization commands, and use the command examples.

Using the GRTMQMAUT command

If you have the required authorization, you can use the GRTMQMAUT command to grant authorization of a user profile or user group to access a particular object. The following examples illustrate how the GRTMQMAUT command is used:

1.
```
GRTMQMAUT OBJ(RED.LOCAL.QUEUE) OBJTYPE(*LCLQ) USER(GROUPA) +
          AUT(*BROWSE *PUT) MQMNAME('saturn.queue.manager')
```

   In this example:

   - RED.LOCAL.QUEUE is the object name.
   - *LCLQ (local queue) is the object type.
   - GROUPA is the name of a user profile on the system for which authorizations are to change. This profile can be used as a group profile for other users.
   - *BROWSE and *PUT are the authorizations being granted to the specified queue.

     *BROWSE adds authorization to browse messages on the queue (to issue MQGET with the browse option).

     *PUT adds authorization to put (MQPUT) messages on the queue.
   - `saturn.queue.manager` is the queue manager name.

2. The following command grants to users JACK and JILL all applicable authorizations, to all process definitions, for the default queue manager.

```
GRTMQMAUT OBJ(*ALL) OBJTYPE(*PRC) USER(JACK JILL) AUT(*ALL)
```

3. The following command grants user GEORGE authority to put a message on the queue ORDERS, on the queue manager TRENT.

   ```
   GRTMQMAUT OBJ(TRENT) OBJTYPE(*MQM) USER(GEORGE) AUT(*CONNECT) MQMNAME (TRENT)
   GRTMQMAUT OBJ(ORDERS) OBJTYPE(*Q) USER(GEORGE) AUT(*PUT) MQMNAME (TRENT)
   ```

Using the RVKMQMAUT command

If you have the required authorization, you can use the RVKMQMAUT command to remove previously granted authorization of a user profile or user group to access a particular object. The following examples illustrate how the RVKMQMAUT command is used:

1.
   ```
   RVKMQMAUT OBJ(RED.LOCAL.QUEUE) OBJTYPE(*LCLQ) USER(GROUPA) +
   AUT(*PUT) MQMNAME('saturn.queue.manager')
   ```

   The authority to put messages to the specified queue, that was granted in the previous example, is removed for GROUPA.
2.
   ```
   RVKMQMAUT OBJ(PAY*) OBJTYPE(*Q) USER(*PUBLIC) AUT(*GET) +
   MQMNAME(PAYROLLQM)
   ```

   Authority to get messages from any queue with a name starting with the characters PAY, owned by queue manager PAYROLLQM, is removed from all users of the system unless they, or a group to which they belong, have been separately authorized.

Using the DSPMQMAUT command

The display MQM authority (DSPMQMAUT) command shows, for the specified object and user, the list of authorizations that the user has for the object. The following example illustrates how the command is used:

```
DSPMQMAUT OBJ(ADMINNL) OBJTYPE(*NMLIST) USER(JOE) OUTPUT(*PRINT) +
MQMNAME(ADMINQM)
```

Using the RFRMQMAUT command

The refresh MQM security (RFRMQMAUT) command enables you to update the OAM's authorization group information immediately, reflecting changes made at the operating system level, without needing to stop and restart the queue manager. The following example illustrates how the command is used:

```
RFRMQMAUT MQMNAME(ADMINQM)
```

IBM i Authorization specification tables on IBM i

Use this information to determine what authorization is required to use particular API calls, and particular options of those calls, on queue objects, process objects, and queue manager objects.

The authorization specification tables starting in Table 19 on page 166 define precisely how the authorizations work and the restrictions that apply. The tables apply to these situations:

- Applications that issue MQI calls
- Administration programs that issue MQSC commands as escape PCFs
- Administration programs that issue PCF commands

In this section, the information is presented as a set of tables that specify the following data:

Action to be performed
MQI option, MQSC command, or PCF command.

Access control object
Queue, process definition, queue manager, namelist, channel, client connection channel, listener, service, or authentication information object.

Authorization required
Expressed as an MQZAO_ constant.

In the tables, the constants prefixed by MQZAO_ correspond to the keywords in the authorization list for the **GRTMQMAUT** and **RVKMQMAUT** commands for the particular entity. For example, MQZAO_BROWSE corresponds to the keyword *BROWSE ; similarly, the keyword MQZAO_SET_ALL_CONTEXT corresponds to the keyword *SETALL, and so on. These constants are defined in the header file cmqzc.h, which is supplied with the product.

MQI authorizations

An application is allowed to issue specific MQI calls and options only if the user identifier under which it is running (or whose authorizations it is able to assume) has been granted the relevant authorization.

Four MQI calls require authorization checks: MQCONN, MQOPEN, MQPUT1, and MQCLOSE.

For MQOPEN and MQPUT1, the authority check is made on the name of the object being opened, and not on the name, or names, resulting after a name has been resolved. For example, an application can be granted authority to open an alias queue without having authority to open the base queue to which the alias resolves. The rule is that the check is carried out on the first definition encountered during the process of name resolution that is not a queue manager alias, unless the queue manager alias definition is opened directly; that is, its name appears in the *ObjectName* field of the object descriptor. Authority is always needed for the particular object being opened; in some cases additional queue-independent authority, obtained through an authorization for the queue manager object, is required.

Table 19 on page 166, Table 20 on page 166, Table 21 on page 167, and Table 22 on page 168 summarize the authorizations needed for each call.

Note: These tables do not mention namelists, channels, client connection channels, listeners, services, or authentication information objects. This is because none of the authorizations apply to these objects, except for MQOO_INQUIRE, for which the same authorizations apply as for the other objects.

Table 19. Security authorization needed for MQCONN calls

Authorization required for:	**Queue object (“1” on page 168)**	**Process object**	**Queue manager object**
MQCONN option	Not applicable	Not applicable	MQZAO_CONNECT

Table 20. Security authorization needed for MQOPEN calls

Authorization required for:	**Queue object (“1” on page 168)**	**Process object**	**Queue manager object**
MQOO_INQUIRE	MQZAO_INQUIRE **(“2” on page 168)**	MQZAO_INQUIRE **(“2” on page 168)**	MQZAO_INQUIRE **(“2” on page 168)**
MQOO_BROWSE	MQZAO_BROWSE	Not applicable	No check
MQOO_INPUT_*	MQZAO_INPUT	Not applicable	No check
MQOO_SAVE_ ALL_CONTEXT **(“3” on page 168)**	MQZAO_INPUT	Not applicable	Not applicable

Table 20. Security authorization needed for MQOPEN calls (continued)

Authorization required for:	**Queue object (“1” on page 168)**	**Process object**	**Queue manager object**
MQOO_OUTPUT (Normal queue) (**“4” on page 168**)	MQZAO_OUTPUT	Not applicable	Not applicable
MQOO_PASS_ IDENTITY_CONTEXT (**“5” on page 168**)	MQZAO_PASS_ IDENTITY_CONTEXT	Not applicable	No check
MQOO_PASS_ALL_ CONTEXT (**“5” on page 168, “6” on page 168**)	MQZAO_PASS _ALL_CONTEXT	Not applicable	No check
MQOO_SET_ IDENTITY_CONTEXT (**“5” on page 168, “6” on page 168**)	MQZAO_SET_ IDENTITY_CONTEXT	Not applicable	MQZAO_SET_ IDENTITY_CONTEXT (**“7” on page 168**)
MQOO_SET_ ALL_CONTEXT (**“5” on page 168, “8” on page 168**)	MQZAO_SET_ ALL_CONTEXT	Not applicable	MQZAO_SET_ ALL_CONTEXT (**“7” on page 168**)
MQOO_OUTPUT (Transmission queue) (**“9” on page 168**)	MQZAO_SET_ ALL_CONTEXT	Not applicable	MQZAO_SET_ ALL_CONTEXT (**“7” on page 168**)
MQOO_SET	MQZAO_SET	Not applicable	No check
MQOO_ALTERNATE_ USER_AUTHORITY	(**“10” on page 168**)	(**“10” on page 168**)	MQZAO_ALTERNATE_ USER_AUTHORITY (**“10” on page 168, “11” on page 168**)

Table 21. Security authorization needed for MQPUT1 calls

Authorization required for:	**Queue object (“1” on page 168)**	**Process object**	**Queue manager object**
MQPMO_PASS_ IDENTITY_CONTEXT	MQZAO_PASS_ IDENTITY_CONTEXT (**“12” on page 168**)	Not applicable	No check
MQPMO_PASS_ALL _CONTEXT	MQZAO_PASS_ ALL_CONTEXT (**“12” on page 168**)	Not applicable	No check
MQPMO_SET_ IDENTITY_CONTEXT	MQZAO_SET_ IDENTITY_CONTEXT (**“12” on page 168**)	Not applicable	MQZAO_SET_ IDENTITY_CONTEXT (**“7” on page 168**)
MQPMO_SET_ ALL_CONTEXT	MQZAO_SET_ ALL_CONTEXT (**“12” on page 168**)	Not applicable	MQZAO_SET_ ALL_CONTEXT (**“7” on page 168**)
(Transmission queue) (**“9” on page 168**)	MQZAO_SET_ ALL_CONTEXT	Not applicable	MQZAO_SET_ ALL_CONTEXT (**“7” on page 168**)

Table 21. Security authorization needed for MQPUT1 calls (continued)

Authorization required for:	Queue object (“1” on page 168)	Process object	Queue manager object
MQPMO_ALTERNATE_ USER_AUTHORITY	**(“13” on page 168)**	Not applicable	MQZAO_ALTERNATE_ USER_AUTHORITY **(“11” on page 168)**

Table 22. Security authorization needed for MQCLOSE calls

Authorization required for:	Queue object (“1” on page 168)	Process object	Queue manager object
MQCO_DELETE	MQZAO_DELETE **(“14” on page 168)**	Not applicable	Not applicable
MQCO_DELETE _PURGE	MQZAO_DELETE **(“14” on page 168)**	Not applicable	Not applicable

Notes for the tables:

1. If a model queue is being opened:
   - MQZAO_DISPLAY authority is needed for the model queue, in addition to the authority to open the model queue for the type of access for which you are opening.
   - MQZAO_CREATE authority is not needed to create the dynamic queue.
   - The user identifier used to open the model queue is automatically granted all the queue-specific authorities (equivalent to MQZAO_ALL) for the dynamic queue created.
2. Either the queue, process, namelist, or queue manager object is checked, depending on the type of object being opened.
3. MQOO_INPUT_* must also be specified. This option is valid for a local, model, or alias queue.
4. This check is performed for all output cases, except the case specified in note “9” on page 168.
5. MQOO_OUTPUT must also be specified.
6. MQOO_PASS_IDENTITY_CONTEXT is also implied by this option.
7. This authority is required for both the queue manager object and the particular queue.
8. MQOO_PASS_IDENTITY_CONTEXT, MQOO_PASS_ALL_CONTEXT, and MQOO_SET_IDENTITY_CONTEXT are also implied by this option.
9. This check is performed for a local or model queue that has a *Usage* queue attribute of MQUS_TRANSMISSION, and is being opened directly for output. It does not apply if a remote queue is being opened (either by specifying the names of the remote queue manager and remote queue, or by specifying the name of a local definition of the remote queue).
10. At least one of MQOO_INQUIRE (for any object type), or (for queues) MQOO_BROWSE, MQOO_INPUT_*, MQOO_OUTPUT, or MQOO_SET must also be specified. The check carried out is as for the other options specified, using the supplied alternate-user identifier for the specific-named object authority, and the current application authority for the MQZAO_ALTERNATE_USER_IDENTIFIER check.
11. This authorization allows any *AlternateUserId* to be specified.
12. An MQZAO_OUTPUT check is also carried out if the queue does not have a *Usage* queue attribute of MQUS_TRANSMISSION.
13. The check carried out is as for the other options specified, using the supplied alternate-user identifier for the named queue authority, and the current application authority for the MQZAO_ALTERNATE_USER_IDENTIFIER check.
14. The check is carried out only if both of the following statements are true:

- A permanent dynamic queue is being closed and deleted.
- The queue was not created by the MQOPEN that returned the object handle being used.

Otherwise, there is no check.

General notes:

1. The special authorization MQZAO_ALL_MQI includes all the following authorizations that are relevant to the object type:
   - MQZAO_CONNECT
   - MQZAO_INQUIRE
   - MQZAO_SET
   - MQZAO_BROWSE
   - MQZAO_INPUT
   - MQZAO_OUTPUT
   - MQZAO_PASS_IDENTITY_CONTEXT
   - MQZAO_PASS_ALL_CONTEXT
   - MQZAO_SET_IDENTITY_CONTEXT
   - MQZAO_SET_ALL_CONTEXT
   - MQZAO_ALTERNATE_USER_AUTHORITY
2. MQZAO_DELETE (see note “14” on page 168) and MQZAO_DISPLAY are classed as administration authorizations. They are not therefore included in MQZAO_ALL_MQI.
3. *No check* means that no authorization checking is carried out.
4. *Not applicable* means that authorization checking is not relevant to this operation. For example, you cannot issue an MQPUT call to a process object.

IBM i *Authorizations for MQSC commands in escape PCFs on IBM i*

These authorizations allow a user to issue administration commands as an escape PCF message. These methods allow a program to send an administration command as a message to a queue manager, for execution on behalf of that user.

This section summarizes the authorizations needed for each MQSC command contained in Escape PCF.

Not applicable means that authorization checking is not relevant to this operation.

The user ID under which the program that submits the command is running must also have the following authorities:

- MQZAO_CONNECT authority to the queue manager
- DISPLAY authority on the queue manager in order to perform PCF commands
- Authority to issue the MQSC commands within the text of the Escape PCF command

ALTER *object*

Object	Authorization required
Queue	MQZAO_CHANGE
Topic	MQZAO_CHANGE
Process	MQZAO_CHANGE
Queue manager	MQZAO_CHANGE
Namelist	MQZAO_CHANGE
Authentication information	MQZAO_CHANGE

Object	Authorization required
Channel	MQZAO_CHANGE
Client connection channel	MQZAO_CHANGE
Listener	MQZAO_CHANGE
Service	MQZAO_CHANGE

CLEAR *object*

Object	Authorization required
Queue	MQZAO_CLEAR
Topic	MQZAO_CLEAR
Process	Not applicable
Queue manager	Not applicable
Namelist	Not applicable
Authentication information	Not applicable
Channel	Not applicable
Client connection channel	Not applicable
Listener	Not applicable
Service	Not applicable

DEFINE *object* NOREPLACE (“1” on page 173)

Object	Authorization required
Queue	MQZAO_CREATE **(“2” on page 173)**
Topic	MQZAO_CREATE **(“2” on page 173)**
Process	MQZAO_CREATE **(“2” on page 173)**
Queue manager	Not applicable
Namelist	MQZAO_CREATE **(“2” on page 173)**
Authentication information	MQZAO_CREATE **(“2” on page 173)**
Channel	MQZAO_CREATE **(“2” on page 173)**
Client connection channel	MQZAO_CREATE **(“2” on page 173)**
Listener	MQZAO_CREATE **(“2” on page 173)**
Service	MQZAO_CREATE **(“2” on page 173)**

DEFINE *object* REPLACE (“1” on page 173, “3” on page 173)

Object	Authorization required
Queue	MQZAO_CHANGE
Topic	MQZAO_CHANGE
Process	MQZAO_CHANGE
Queue manager	Not applicable

Object	Authorization required
Namelist	MQZAO_CHANGE
Authentication information	MQZAO_CHANGE
Channel	MQZAO_CHANGE
Client connection channel	MQZAO_CHANGE
Listener	MQZAO_CHANGE
Service	MQZAO_CHANGE

DELETE *object*

Object	Authorization required
Queue	MQZAO_DELETE
Topic	MQZAO_DELETE
Process	MQZAO_DELETE
Queue manager	Not applicable
Namelist	MQZAO_DELETE
Authentication information	MQZAO_DELETE
Channel	MQZAO_DELETE
Client connection channel	MQZAO_DELETE
Listener	MQZAO_DELETE
Service	MQZAO_DELETE

DISPLAY *object*

Object	Authorization required
Queue	MQZAO_DISPLAY
Topic	MQZAO_DISPLAY
Process	MQZAO_DISPLAY
Queue manager	MQZAO_DISPLAY
Namelist	MQZAO_DISPLAY
Authentication information	MQZAO_DISPLAY
Channel	MQZAO_DISPLAY
Client connection channel	MQZAO_DISPLAY
Listener	
Service	

PING CHANNEL

Object	Authorization required
Queue	Not applicable
Topic	Not applicable

Object	Authorization required
Process	Not applicable
Queue manager	Not applicable
Namelist	Not applicable
Authentication information	Not applicable
Channel	MQZAO_CONTROL
Client connection channel	Not applicable
Listener	Not applicable
Service	Not applicable

RESET CHANNEL

Object	Authorization required
Queue	Not applicable
Topic	Not applicable
Process	Not applicable
Queue manager	Not applicable
Namelist	Not applicable
Authentication information	Not applicable
Channel	MQZAO_CONTROL_EXTENDED
Client connection channel	Not applicable
Listener	Not applicable
Service	Not applicable

RESOLVE CHANNEL

Object	Authorization required
Queue	Not applicable
Topic	Not applicable
Process	Not applicable
Queue manager	Not applicable
Namelist	Not applicable
Authentication information	Not applicable
Channel	MQZAO_CONTROL_EXTENDED
Client connection channel	Not applicable
Listener	Not applicable
Service	Not applicable

START ***object***

Object	Authorization required
Queue	Not applicable
Topic	Not applicable
Process	Not applicable
Queue manager	Not applicable
Namelist	Not applicable
Authentication information	Not applicable
Channel	MQZAO_CONTROL
Client connection channel	Not applicable
Listener	MQZAO_CONTROL
Service	MQZAO_CONTROL

STOP ***object***

Object	Authorization required
Queue	Not applicable
Topic	Not applicable
Process	Not applicable
Queue manager	Not applicable
Namelist	Not applicable
Authentication information	Not applicable
Channel	MQZAO_CONTROL
Client connection channel	Not applicable
Listener	MQZAO_CONTROL
Service	MQZAO_CONTROL

Note:

1. For DEFINE commands, MQZAO_DISPLAY authority is also needed for the LIKE object if one is specified, or on the appropriate SYSTEM.DEFAULT.xxx object if LIKE is omitted.
2. The MQZAO_CREATE authority is not specific to a particular object or object type. Create authority is granted for all objects for a specified queue manager, by specifying an object type of QMGR on the `GRTMQMAUT` command.
3. This option applies if the object to be replaced already exists. If it does not, the check is as for DEFINE *object* NOREPLACE.

IBM i *Authorizations for PCF commands on IBM i*

These authorizations allow a user to issue administration commands as PCF commands. These methods allow a program to send an administration command as a message to a queue manager, for execution on behalf of that user.

This section summarizes the authorizations needed for each PCF command.

No check means that no authorization checking is carried out; *Not applicable* means that authorization checking is not relevant to this operation.

The user ID under which the program that submits the command is running must also have the following authorities:

- MQZAO_CONNECT authority to the queue manager
- DISPLAY authority on the queue manager in order to perform PCF commands

The special authorization MQZAO_ALL_ADMIN includes the following authorizations:

- MQZAO_CHANGE
- MQZAO_CLEAR
- MQZAO_DELETE
- MQZAO_DISPLAY
- MQZAO_CONTROL
- MQZAO_CONTROL_EXTENDED

MQZAO_CREATE is not included as it is not specific to a particular object or object type

Change *object*

Object	**Authorization required**
Queue	MQZAO_CHANGE
Topic	MQZAO_CHANGE
Process	MQZAO_CHANGE
Queue manager	MQZAO_CHANGE
Namelist	MQZAO_CHANGE
Authentication information	MQZAO_CHANGE
Channel	MQZAO_CHANGE
Client connection channel	MQZAO_CHANGE
Listener	MQZAO_CHANGE
Service	MQZAO_CHANGE

Clear *object*

Object	**Authorization required**
Queue	MQZAO_CLEAR
Topic	MQZAO_CLEAR
Process	Not applicable
Queue manager	Not applicable
Namelist	Not applicable
Authentication information	Not applicable
Channel	Not applicable
Client connection channel	Not applicable
Listener	Not applicable
Service	Not applicable

Copy *object* (without replace) (“1” on page 179)

Object	Authorization required
Queue	MQZAO_CREATE **(“2” on page 179)**
Topic	MQZAO_CREATE **(“2” on page 179)**
Process	MQZAO_CREATE **(“2” on page 179)**
Queue manager	Not applicable
NamelistMQZAO_CREATE	MQZAO_CREATE **(“2” on page 179)**
Authentication information	MQZAO_CREATE **(“2” on page 179)**
Channel	MQZAO_CREATE **(“2” on page 179)**
Client connection channel	MQZAO_CREATE **(“2” on page 179)**
Listener	MQZAO_CREATE **(“2” on page 179)**
Service	MQZAO_CREATE **(“2” on page 179)**

Copy *object* (with replace) (“1” on page 179, “4” on page 179)

Object	Authorization required
Queue	MQZAO_CHANGE
Topic	MQZAO_CHANGE
Process	MQZAO_CHANGE
Queue manager	Not applicable
Namelist	MQZAO_CHANGE
Authentication information	MQZAO_CHANGE
Channel	MQZAO_CHANGE
Client connection channel	MQZAO_CHANGE
Listener	MQZAO_CHANGE
Service	MQZAO_CHANGE

Create *object* (without replace) (“3” on page 179)

Object	Authorization required
Queue	MQZAO_CREATE **(“2” on page 179)**
Topic	MQZAO_CREATE **(“2” on page 179)**
Process	MQZAO_CREATE **(“2” on page 179)**
Queue manager	Not applicable
Namelist	MQZAO_CREATE **(“2” on page 179)**
Authentication information	MQZAO_CREATE **(“2” on page 179)**
Channel	MQZAO_CREATE **(“2” on page 179)**
Client connection channel	MQZAO_CREATE **(“2” on page 179)**
Listener	MQZAO_CHANGE

Object	Authorization required
Service	MQZAO_CHANGE

Create *object* (with replace) (“3” on page 179, “4” on page 179)

Object	Authorization required
Queue	MQZAO_CHANGE
Topic	MQZAO_CHANGE
Process	MQZAO_CHANGE
Queue manager	Not applicable
Namelist	MQZAO_CHANGE
Authentication information	MQZAO_CHANGE
Channel	MQZAO_CHANGE
Client connection channel	MQZAO_CHANGE
Listener	MQZAO_CHANGE
Service	MQZAO_CHANGE

Delete *object*

Object	Authorization required
Queue	MQZAO_DELETE
Topic	MQZAO_DELETE
Process	MQZAO_DELETE
Queue manager	MQZAO_DELETE
Namelist	MQZAO_DELETE
Authentication information	MQZAO_DELETE
Channel	MQZAO_DELETE
Client connection channel	MQZAO_DELETE
Listener	MQZAO_DELETE
Service	MQZAO_DELETE

Inquire *object*

Object	Authorization required
Queue	MQZAO_DISPLAY
Topic	MQZAO_DISPLAY
Process	MQZAO_DISPLAY
Queue manager	MQZAO_DISPLAY
Namelist	MQZAO_DISPLAY
Authentication information	MQZAO_DISPLAY
Channel	MQZAO_DISPLAY

Object	Authorization required
Client connection channel	MQZAO_DISPLAY
Listener	MQZAO_DISPLAY
Service	MQZAO_DISPLAY

Inquire *object* names

Object	Authorization required
Queue	No check
Topic	No check
Process	No check
Queue manager	No check
Namelist	No check
Authentication information	No check
Channel	No check
Client connection channel	No check
Listener	No check
Service	No check

Ping Channel

Object	Authorization required
Queue	Not applicable
Topic	Not applicable
Process	Not applicable
Queue manager	Not applicable
Namelist	Not applicable
Authentication information	Not applicable
Channel	MQZAO_CONTROL
Client connection channel	Not applicable
Listener	Not applicable
Service	Not applicable

Reset Channel

Object	Authorization required
Queue	Not applicable
Topic	Not applicable
Process	Not applicable
Queue manager	Not applicable
Namelist	Not applicable

Object	Authorization required
Authentication information	Not applicable
Channel	MQZAO_CONTROL_EXTENDED
Client connection channel	Not applicable
Listener	Not applicable
Service	Not applicable

Reset Queue Statistics

Object	Authorization required
Queue	MQZAO_DISPLAY and MQZAO_CHANGE
Topic	Not applicable
Process	Not applicable
Queue manager	Not applicable
Namelist	Not applicable
Authentication information	Not applicable
Channel	Not applicable
Client connection channel	Not applicable
Listener	
Service	

Resolve Channel

Object	Authorization required
Queue	Not applicable
Topic	Not applicable
Process	Not applicable
Queue manager	Not applicable
Namelist	Not applicable
Authentication information	Not applicable
Channel	MQZAO_CONTROL_EXTENDED
Client connection channel	Not applicable
Listener	Not applicable
Service	Not applicable

Start Channel

Object	Authorization required
Queue	Not applicable
Topic	Not applicable
Process	Not applicable

Object	Authorization required
Queue manager	Not applicable
Namelist	Not applicable
Authentication information	Not applicable
Channel	MQZAO_CONTROL
Client connection channel	Not applicable
Listener	Not applicable
Service	Not applicable

Stop Channel

Object	Authorization required
Queue	Not applicable
Topic	Not applicable
Process	Not applicable
Queue manager	Not applicable
Namelist	Not applicable
Authentication information	Not applicable
Channel	MQZAO_CONTROL
Client connection channel	Not applicable
Listener	Not applicable
Service	Not applicable

Note:

1. For Copy commands, MQZAO_DISPLAY authority is also needed for the From object.
2. The MQZAO_CREATE authority is not specific to a particular object or object type. Create authority is granted for all objects for a specified queue manager, by specifying an object type of QMGR on the `GRTMQMAUT` command.
3. For Create commands, MQZAO_DISPLAY authority is also needed for the appropriate SYSTEM.DEFAULT.* object.
4. This option applies if the object to be replaced already exists. If it does not, the check is as for Copy or Create without replace.

IBM i Generic OAM profiles on IBM i

Object authority manager (OAM) generic profiles enable you to set the authority a user has to many objects at once, rather than having to issue separate **`GRTMQMAUT`** commands against each individual object when it is created. Using generic profiles in the **`GRTMQMAUT`** command enables you to set a generic authority for all future objects created that fit that profile.

The rest of this section describes the use of generic profiles in more detail:

- "Using wildcard characters" on page 180
- "Profile priorities" on page 180

Using wildcard characters

What makes a profile generic is the use of special characters (wildcard characters) in the profile name. For example, the question mark (?) wildcard character matches any single character in a name. So, if you specify `ABC.?EF`, the authorization you give to that profile applies to any objects created with the names `ABC.DEF`, `ABC.CEF`, `ABC.BEF`, and so on.

The wildcard characters available are:

?

Use the question mark (?) instead of any single character. For example, `AB.?D` would apply to the objects `AB.CD`, `AB.ED`, and `AB.FD`.

*****

Use the asterisk (*) as:

- A *qualifier* in a profile name to match any one qualifier in an object name. A qualifier is the part of an object name delimited by a period. For example, in `ABC.DEF.GHI`, the qualifiers are `ABC`, `DEF`, and `GHI`.

  For example, `ABC.*.JKL` would apply to the objects `ABC.DEF.JKL`, and `ABC.GHI.JKL`. (Note that it would **not** apply to `ABC.JKL` ; * used in this context always indicates one qualifier.)
- A character within a qualifier in a profile name to match zero or more characters within the qualifier in an object name.

  For example, `ABC.DE*.JKL` would apply to the objects `ABC.DE.JKL`, `ABC.DEF.JKL`, and `ABC.DEGH.JKL`.

******

Use the double asterisk (**) ***once*** in a profile name as:

- The entire profile name to match all object names. For example, if you use the keyword `OBJTYPE (*PRC)` to identify processes, then use ** as the profile name, you change the authorizations for all processes.
- As either the beginning, middle, or ending qualifier in a profile name to match zero or more qualifiers in an object name. For example, `**.ABC` identifies all objects with the final qualifier ABC.

Profile priorities

An important point to understand when using generic profiles is the priority that profiles are given when deciding what authorities to apply to an object being created. For example, suppose that you have issued the commands:

```
GRTMQMAUT OBJ(AB.*) OBJTYPE(*Q) USER(FRED) AUT(*PUT) MQMNAME(MYQMGR)
GRTMQMAUT OBJ(AB.C*) OBJTYPE(*Q) USER(FRED) AUT(*GET) MQMNAME(MYQMGR)
```

The first gives put authority to all queues for the principal FRED with names that match the profile AB.*; the second gives get authority to the same types of queue that match the profile AB.C*.

Suppose that you now create a queue called AB.CD. According to the rules for wildcard matching, either GRTMQMAUT could apply to that queue. So, does it have put or get authority?

To find the answer, you apply the rule that, whenever multiple profiles can apply to an object, **only the most specific applies**. The way that you apply this rule is by comparing the profile names from left to right. Wherever they differ, a non-generic character is more specific than a generic character. So, in the previous example, the queue AB.CD has **get** authority (AB.C* is more specific than AB.*).

When you are comparing generic characters, the order of *specificity* is:

1. ?
2. *
3. **

IBM i Specifying the installed authorization service on IBM i

You can specify which authorization service component to use.

The parameter **`Service Component name`** on **GRTMQMAUT** and **RVKMQMAUT** allows you to specify the name of the installed authorization service component.

Selecting **F24** on the initial panel, followed by **F9=All parameters** on the next panel of either command, allows you to specify either the installed authorization component (*DFT) or the name of the required authorization service component specified in the Service stanza of the queue manager's qm.ini file.

DSPMQMAUT also has this extra parameter. This parameter allows you to search all the installed authorization components (*DFT), or the specified authorization-service component name, for the specified object name, object type, and user

IBM i Working with and without authority profiles on IBM i

Use this information to learn how to work with authority profiles and how to work without authority profiles.

You can work with authority profiles, as explained in "Working with authority profiles" on page 181, or without them, as explained here:

To work without authority profiles, use `*NONE` as an Authority parameter on **GRTMQMAUT** to create profiles without authority. This leaves any existing profiles unchanged.

On **RVKMQMAUT**, use `*REMOVE` as an Authority parameter to remove an existing authority profile.

Working with authority profiles

There are two commands associated with authority profiling:

- **WRKMQMAUT**
- **WRKMQMAUTD**

You can access these commands directly from the command line, or from the WRKMQM panel by:

1. Typing in the queue manager name and pressing the `Enter` key to access the **WRKMQM** results panel.
2. Selecting `F23=More options` on this panel.

Option 24 selects the results panel for the **WRKMQMAUT** command and option 25 selects the **WRKMQMAUTI** command, which is used with the SSL bindings layer.

WRKMQMAUT

This command allows you to work with the authority data held in the authority queue.

Note: To run this command you must have `*connect` and `*admdsp` authority to the queue manager. However, to create or delete a profile, you need QMQMADM authority.

If you output the information to the screen, a list of authority profile names, together with their types, is displayed. If you print the output, you receive a detailed list of all the authority data, the registered users, and their authorities.

Entering an object or profile name on this panel, and pressing ENTER takes you to the results panel for **WRKMQMAUT** .

If you select `4=Delete`, you go to a new panel from which you can confirm that you want to delete all the user names registered to the generic authority profile name you specify. This option runs **RVKMQMAUT** with the option *REMOVE for all the users, and applies **only** to generic profile names.

If you select `12=Work with profile` you go to the **WRKMQMAUTD** command results panel, as explained in "WRKMQMAUTD" on page 182.

WRKMQMAUTD

This command allows you to display all the users registered with a particular authority profile name and object type. To run this command you must have `*connect` and `*admdsp` authority to the queue manager. However, to grant, run, create, or delete a profile you need QMQMADM authority.

Selecting `F24=More keys` from the initial input panel, followed by option `F9=All Parameters` displays the Service Component Name as for **GRTMQMAUT** and **RVKMQMAUT**.

Note: The `F11=Display Object Authorizations` key toggles between the following types of authorities:

- Object authorizations
- Context authorizations
- MQI authorizations

The options on the screen are:

2=Grant
Takes you to the **GRTMQMAUT** panel to add to the current authorities.

3=Revoke
Takes you to the **RVKMQMAUT** panel to remove some of the current definitions

4=Delete
Takes you to a panel that allows you to delete the authority data for specified users. This runs **RVKMQMAUT** with the option *REMOVE.

5=Display
Takes you to the existing **DSPMQMAUT** command

F6=Create
Takes you to the **GRTMQMAUT** panel that allows you to create a profile authority record.

IBM i Object Authority Manager guidelines on IBM i

Additional hints and tips for using the object authority manager (OAM)

Limit access to sensitive operations

Some operations are sensitive; limit them to privileged users. For example,

- Accessing some special queues, such as transmission queues or the command queue `SYSTEM.ADMIN.COMMAND.QUEUE`
- Running programs that use full MQI context options
- Creating and copying application queues

Queue manager directories

The directories and libraries containing queues and other queue manager data are private to the product. Do not use standard operating system commands to grant or revoke authorizations to MQI resources.

Queues

The authority to a dynamic queue is based on, but is not necessarily the same as, that of the model queue from which it is derived.

For alias queues and remote queues, the authorization is that of the object itself, not the queue to which the alias or remote queue resolves. It is possible to authorize a user profile to access an alias queue that resolves to a local queue to which the user profile has no access permissions.

Limit the authority to create queues to privileged users. If you do not, users can bypass the normal access control by creating an alias.

Alternate-user authority

Alternate-user authority controls whether one user profile can use the authority of another user profile when accessing an IBM MQ object. This technique is essential where a server receives requests from a program and the server wants to ensure that the program has the required authority for the request. The server might have the required authority, but it needs to know whether the program has the authority for the actions it has requested.

For example:

- A server program running under user profile PAYSERV retrieves a request message from a queue that was put on the queue by user profile USER1.
- When the server program gets the request message, it processes the request and puts the reply back into the reply-to queue specified with the request message.
- Instead of using its own user profile (PAYSERV) to authorize opening the reply-to queue, the server can specify some other user profile, in this case, USER1. In this example, you can use alternate-user authority to control whether PAYSERV is allowed to specify USER1 as an alternate-user profile when it opens the reply-to queue.

The alternate-user profile is specified on the *AlternateUserId* field of the object descriptor.

Note: You can use alternate-user profiles on any IBM MQ object. Use of an alternate-user profile does not affect the user profile used by any other resource managers.

Context authority

Context is information that applies to a particular message and is contained in the message descriptor, MQMD, which is part of the message.

For descriptions of the message descriptor fields relating to context, see MQMD - Message descriptor.

For information about the context options, see Message context.

Remote security considerations

For remote security, consider:

Put authority
For security across queue managers, you can specify the put authority that is used when a channel receives a message sent from another queue manager.

This parameter is valid only for RCVR, RQSTR, or CLUSRCVR channel types. Specify the channel attribute PUTAUT as follows:

DEF
Default user profile. This is the QMQM user profile under which the message channel agent is running.

CTX
The user profile in the message context.

Transmission queues
Queue managers automatically put remote messages on a transmission queue; no special authority is required. However, putting a message directly on a transmission queue requires special authorization.

Channel exits
Channel exits can be used for added security.

Channel authentication records
Use to exercise more precise control over the access granted to connecting systems at a channel level.

For more information about remote security, see “Channel authorization” on page 113.

Protecting channels with SSL/TLS

The Transport Layer Security (TLS) protocol provide channel security, with protection against eavesdropping, tampering, and impersonation. IBM MQ support for TLS enables you to specify, on the channel definition, that a particular channel uses TLS security. You can also specify details of the security you want, such as the encryption algorithm you want to use.

TLS support in IBM MQ uses the queue manager *authentication information object* and various CL and MQSC commands and queue manager and channel parameters that define the TLS support required in detail.

The following CL commands support TLS:

WRKMQMAUTI
Work with the attributes of an authentication information object.

CHGMQMAUTI
Modify the attributes of an authentication information object.

CRTMQMAUTI
Create an authentication information object.

CPYMQMAUTI
Create an authentication information object by copying an existing one.

DLTMQMAUTI
Delete an authentication information object.

DSPMQMAUTI
Displays the attributes for a specific authentication information object.

For an overview of channel security using TLS, see

- Protecting channels with TLS

For details of PCF commands associated with TLS, see

- Change, Copy, and Create Authentication Information Object
- Delete Authentication Information Object
- Inquire Authentication Information Object

z/OS Setting up security on z/OS

Security considerations specific to z/OS.

Security in IBM MQ for z/OS is controlled using RACF or an equivalent external security manager (ESM).

The following instructions assume that you are using RACF.

Related concepts
Security scenario: two queue managers on z/OS
Security scenario: queue sharing group on z/OS

z/OS RACF security classes

RACF classes are used to hold the profiles required for IBM MQ security checking. Many of the member classes have equivalent group classes. You must activate the classes and enable them to accept generic profiles.

Each RACF class holds one or more profiles used at some point in the checking sequence, as shown in Table 23 on page 185.

Table 23. RACF classes used by IBM MQ

Member class	**Group class**	**Contents**
MQADMIN	GMQADMIN	Profiles that are used mainly for administrative functions. For example: • Profiles for IBM MQ security switches. • The RESLEVEL security profile. • Profiles for alternate user security. • Profiles for context security. • Profiles for command resource security. This class can hold only uppercase RACF profiles.
MXADMIN	GMXADMIN	Profiles that are used mainly for administrative functions. For example: • Profiles for IBM MQ security switches. • The RESLEVEL security profile. • Profiles for alternate user security. • Profiles for context security. • Profiles for command resource security. This class can hold both uppercase and mixed-case RACF profiles.
MQCONN		Profiles used for connection security.
MQCMDS		Profiles used for command security.
MQQUEUE	GMQQUEUE	Uppercase profiles used in queue resource security.
MXQUEUE	GMXQUEUE	Mixed-case and uppercase profiles used in queue resource security.
MQPROC	GMQPROC	Uppercase profiles used in process resource security.
MXPROC	GMXPROC	Mixed-case and uppercase profiles used in process resource security.
MQNLIST	GMQNLIST	Uppercase profiles used in namelist resource security.
MXNLIST	GMXNLIST	Mixed-case and uppercase profiles used in namelist resource security.
MXTOPIC	GMXTOPIC	Mixed-case and uppercase profiles used in topic security.

Some classes have a related *group class* that enables you to put together groups of resources that have similar access requirements. For details about the difference between the member and group classes and when to use a member or group class, see the z/OS Security Server RACF Security Administrator's Guide.

The classes must be activated before security checks can be made. To activate all the IBM MQ classes, you can use this RACF command:

```
SETROPTS CLASSACT(MQADMIN,MXADMIN,MQQUEUE,MXQUEUE,MQPROC,MXPROC,
                  MQNLIST,MXNLIST,MXTOPIC,MQCONN,MQCMDS)
```

You should also ensure that you set up the classes so that they can accept generic profiles. You also do this with the RACF command **SETROPTS**, for example:

```
SETROPTS GENERIC(MQADMIN,MXADMIN,MQQUEUE,MXQUEUE,MQPROC,MXPROC,
                 MQNLIST,MXNLIST,MXTOPIC,MQCONN,MQCMDS)
```

z/OS RACF profiles

All RACF profiles used by IBM MQ contain a prefix, which is either the queue manager name or the queue sharing group name. Be careful when you use the percent sign as a wildcard.

All RACF profiles used by IBM MQ contain a prefix. For queue sharing group level security, this is the queue sharing group name. For queue manager level security, the prefix is the queue manager name. If you are using a mixture of queue manager and queue sharing group level security, you will use profiles with both types of prefix. Queue sharing group and queue manager level security are described in Security controls and options in IBM MQ for z/OS.

For example, if you want to protect a queue called QUEUE_FOR_SUBSCRIBER_LIST in queue sharing group QSG1 at queue sharing group level, the appropriate profile would be defined to RACF as:

```
RDEFINE MQQUEUE QSG1.QUEUE_FOR_SUBSCRIBER_LIST
```

If you want to protect a queue called QUEUE_FOR_LOST_CARD_LIST, that belongs to queue manager STCD at queue manager level, the appropriate profile would be defined to RACF as:

```
RDEFINE MQQUEUE STCD.QUEUE_FOR_LOST_CARD_LIST
```

This means that different queue managers and queue sharing groups can share the same RACF database and yet have different security options.

Do not use generic queue manager names in profiles to avoid unanticipated user access.

IBM MQ allows the use of the percent sign (%) in object names. However, RACF uses the % character as a single-character wildcard. This means that when you define an object name with a % character in its name, you must consider this when you define the corresponding profile.

For example, for the queue CREDIT_CARD_%_RATE_INQUIRY, on queue manager CRDP, the profile would be defined to RACF as follows:

```
RDEFINE MQQUEUE CRDP.CREDIT_CARD_%_RATE_INQUIRY
```

This queue cannot be protected by a generic profile, such as, CRDP.**.

IBM MQ allows the use of mixed-case characters in object names. You can protect these objects by defining:

1. Mixed-case profiles in the appropriate mixed-case RACF classes, or
2. Generic profiles in the appropriate uppercase RACF classes.

To use mixed-case profiles and mixed-case RACF classes you must follow the steps described in "Migrating a z/OS queue manager to mixed-case security" on page 263.

There are some profiles, or parts of profiles, that remain uppercase only as the values are provided by IBM MQ. These are:

- Switch profiles.
- All high-level qualifiers (HLQ) including subsystem and queue sharing group identifiers.
- Profiles for SYSTEM objects.
- Profiles for Default objects.

- The **MQCMDS** class, so all command profiles are uppercase only.
- The **MQCONN** class, so all connection profiles are uppercase only.
- **RESLEVEL** profiles.
- The `'object'` qualification in command resource profiles; for example, `hlq.QUEUE.queuename`. The resource name only is mixed case.
- Dynamic queue profiles `hlq.CSQOREXX.*`, `hlq.CSQUTIL.*`, and `CSQXCMD.*`.
- The `'CONTEXT'` part of `hlq.CONTEXT.resourcename`.
- The `'ALTERNATE.USER'` part of `hlq.ALTERNATE.USER.userid`.

For example, you can define a profile to grant access to a queue called `PAYROLL.Dept1` on queue manager `QM01` in one of the following ways.

- If you are using mixed-case profiles, you can define a profile in the IBM MQ RACF class MXQUEUE using the following command:

  ```
  RDEFINE MXQUEUE MQ01.PAYROLL.Dept1
  ```

- If you are using uppercase profiles, you can define a profile in the IBM MQ RACF class MQQUEUE using the following command:

  ```
  RDEFINE MQQUEUE MQ01.PAYROLL.*
  ```

The first example, using mixed-case profiles, gives you more granular control over granting authority to access the resource.

z/OS Switch profiles

To control the security checking performed by IBM MQ, you use *switch profiles*. A switch profile is a normal RACF profile that has a special meaning to IBM MQ. The access list in switch profiles is not used by IBM MQ.

IBM MQ maintains an internal switch for each switch type shown in tables Switch profiles for subsystem level security, Switch profiles for queue sharing group or queue manager level security,and Switch profiles for resource checking. Switch profiles can be maintained at queue sharing group level, or at queue manager level, or at a combination of both. Using a single set of queue sharing group security switch profiles, you can control security on all the queue managers within a queue sharing group.

When a security switch is set on, the security checks associated with the switch are performed. When a security switch is set off, the security checks associated with the switch are bypassed. The default is that all security switches are set on.

z/OS *Switches and classes*

When you start a queue manager or refresh security, IBM MQ sets switches according to the state of various RACF classes.

When a queue manager is started (or when the MQADMIN or MXADMIN class is refreshed by the IBM MQ REFRESH SECURITY command), IBM MQ first checks the status of RACF and the appropriate class:

- The MQADMIN class if you are using uppercase profiles
- The MXADMIN class if you are using mixed case profile.

It sets the subsystem security switch off if any of these conditions is true:

- RACF is inactive or not installed.
- The MQADMIN or MXADMIN class is not defined (these classes are always defined for RACF because they are included in the class descriptor table (CDT)).
- The MQADMIN or MXADMIN class has not been activated.

If both RACF and the MQADMIN or MXADMIN class are active, IBM MQ checks the MQADMIN or MXADMIN class to see whether any of the switch profiles have been defined. It first checks the profiles described in "Profiles to control subsystem security" on page 188. If subsystem security is not required, IBM MQ sets the internal subsystem security switch off, and performs no further checks.

The profiles determine whether the corresponding IBM MQ switch is set on or off.

- If the switch is off, that type of security is deactivated.
- If any IBM MQ switch is set on, IBM MQ checks the status of the RACF class associated with the type of security corresponding to the IBM MQ switch. If the class is not installed or not active, the IBM MQ switch is set off. For example, process security checks are not carried out if the MQPROC or MXPROC class has not been activated. The class not being active is equivalent to defining NO.PROCESS.CHECKS profile for every queue manager and queue sharing group that uses this RACF database.

z/OS *How switches work*

To set off a security switch, define a NO.* switch profile for it. You can override a NO.* profile set at the queue sharing group level by defining a YES.* profile for a queue manager.

To set off a security switch, you need to define a NO.* switch profile for it. The existence of a NO.* profile means that security checks are **not** performed for that type of resource, unless you choose to override a queue sharing group level setting on a particular queue manager. This is described in "Overriding queue sharing group level settings" on page 188.

If your queue manager is not a member of a queue sharing group, you do not need to define any queue sharing group level profiles or any override profiles. However, you must remember to define these profiles if the queue manager joins a queue sharing group at a later date.

Each NO.* switch profile that IBM MQ detects turns off the checking for that type of resource. Switch profiles are activated during startup of the queue manager. If you change the switch profiles while any affected queue managers are running, you can get IBM MQ to recognize the changes by issuing the IBM MQ REFRESH SECURITY command.

The switch profiles must always be defined in the MQADMIN or MXADMIN class. Do not define them in the GMQADMIN or GMXADMIN class. Tables Switch profiles for subsystem level security and Switch profiles for resource checking show the valid switch profiles and the security type they control.

Overriding queue sharing group level settings

You can override queue sharing group level security settings for a particular queue manager that is a member of that group. If you want to perform queue manager checks on an individual queue manager that are not performed on other queue managers in the group, use the (qmgr-name.YES.*) switch profiles.

Conversely, if you do not want to perform a certain check on one particular queue manager within a queue sharing group, define a (qmgr-name.NO.*) profile for that particular resource type on the queue manager, and do not define a profile for the queue sharing group. (IBM MQ only checks for a queue sharing group level profile if it does not find a queue manager level profile.)

z/OS *Profiles to control subsystem security*

IBM MQ checks whether subsystem security checks are required for the subsystem, for the queue manager, and for the queue sharing group.

The first security check made by IBM MQ is used to determine whether security checks are required for the whole IBM MQ subsystem. If you specify that you do not want subsystem security, no further checks are made.

The following switch profiles are checked to determine whether subsystem security is required. Figure 14 on page 189 shows the order in which they are checked.

Table 24. Switch profiles for subsystem level security

Switch profile name	Type of resource or checking that is controlled
qmgr-name.NO.SUBSYS.SECURITY	Subsystem security for this queue manager
qsg-name.NO.SUBSYS.SECURITY	Subsystem security for this queue sharing group
qmgr-name.YES.SUBSYS.SECURITY	Subsystem security override for this queue manager

If your queue manager is not a member of a queue sharing group, IBM MQ checks for the qmgr-name.NO.SUBSYS.SECURITY switch profile only.

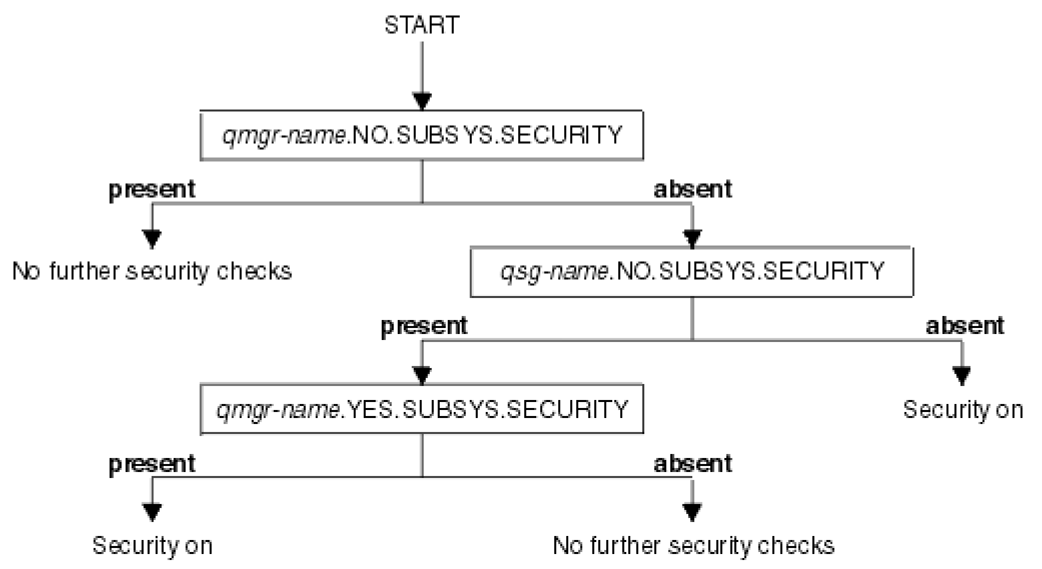

START
qmgr-name.NO.SUBSYS.SECURITY
present
absent
No further security checks
qsg-name.NO.SUBSYS.SECURITY
present
absent
qmgr-name.YES.SUBSYS.SECURITY
Security on
present
absent
Security on
No further security checks

Figure 14. Checking for subsystem security

z/OS Profiles to control queue sharing group or queue manager level security

If subsystem security checking is required, IBM MQ checks whether security checking is required at queue sharing group or queue manager level.

When IBM MQ has determined that security checking is required, it then determines whether checking is required at queue sharing group or queue manager level, or both. These checks are not performed if your queue manager is not a member of a queue sharing group.

The following switch profiles are checked to determine the level required. Figure 15 on page 190 and Figure 16 on page 190 show the order in which they are checked.

Table 25. Switch profiles for queue sharing group or queue manager level security

Switch profile name	Type of resource or checking that is controlled
qmgr-name.NO.QMGR.CHECKS	No queue manager level checks for this queue manager
qsg-name.NO.QMGR.CHECKS	No queue manager level checks for this queue sharing group
qmgr-name.YES.QMGR.CHECKS	Queue manager level checks override for this queue manager
qmgr-name.NO.QSG.CHECKS	No queue sharing group level checks for this queue manager
qsg-name.NO.QSG.CHECKS	No queue sharing group level checks for this queue sharing group

Table 25. Switch profiles for queue sharing group or queue manager level security (continued)

Switch profile name	Type of resource or checking that is controlled
qmgr-name.YES.QSG.CHECKS	Queue sharing group level checks override for this queue manager

If subsystem security is active, you cannot switch off both queue sharing group and queue manager level security. If you try to do so, IBM MQ sets security checking on at both levels.

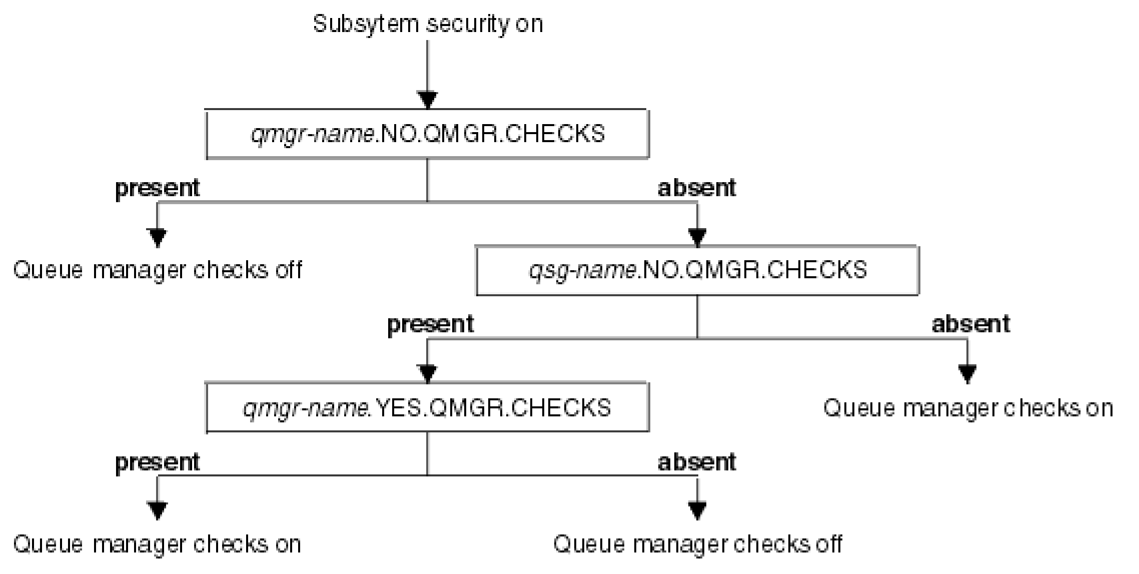


Figure 15. Checking for queue manager level security

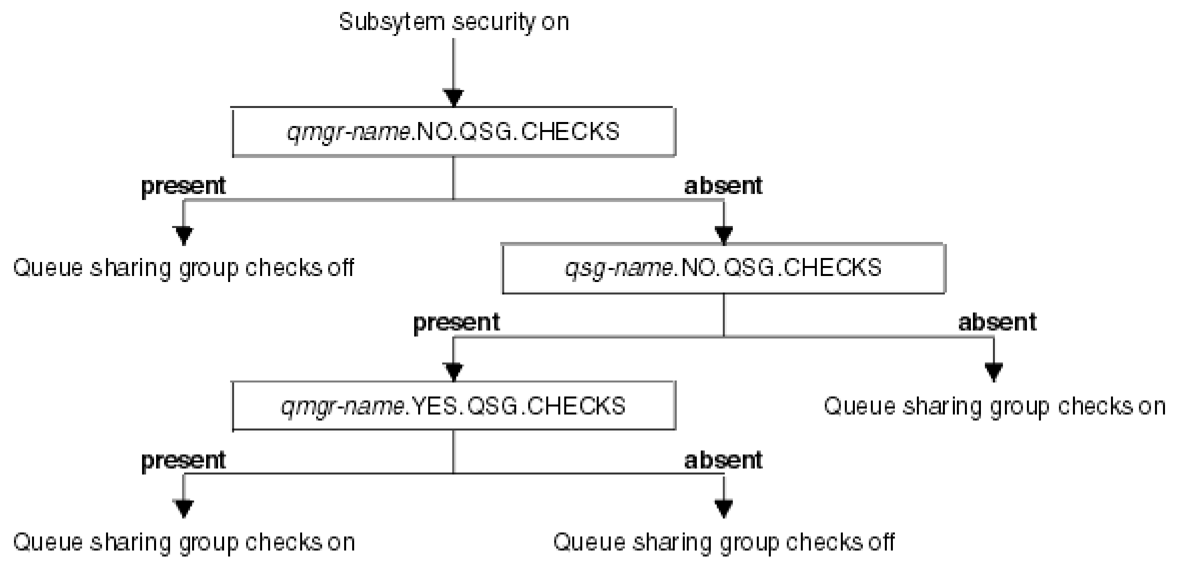


Figure 16. Checking for queue sharing group level security

z/OS *Valid combinations of security switches*

Only certain combinations of switches are valid. If you use a combination of switch settings that is not valid, message CSQH026I is issued and security checking is set on at both queue sharing group and queue manager level.

Table 26 on page 191, Table 27 on page 191, Table 28 on page 191, and Table 29 on page 192 show the sets of combinations of switch settings that are valid for each type of security level.

Table 26. Valid security switch combinations for queue manager level security

Combinations
qmgr-name.NO.QSG.CHECKS
qsg-name.NO.QSG.CHECKS
qmgr-name.NO.QSG.CHECKS qsg-name.NO.QMGR.CHECKS qmgr-name.YES.QMGR.CHECKS
qsg-name.NO.QSG.CHECKS qsg-name.NO.QMGR.CHECKS qmgr-name.YES.QMGR.CHECKS

Table 27. Valid security switch combinations for queue sharing group level security

Combinations
qmgr-name.NO.QMGR.CHECKS
qsg-name.NO.QMGR.CHECKS
qmgr-name.NO.QMGR.CHECKS qsg-name.NO.QSG.CHECKS qmgr-name.YES.QSG.CHECKS
qsg-name.NO.QMGR.CHECKS qsg-name.NO.QSG.CHECKS qmgr-name.YES.QSG.CHECKS

Table 28. Valid security switch combinations for queue manager and queue sharing group level security

Combinations
qsg-name.NO.QMGR.CHECKS qmgr-name.YES.QMGR.CHECKS No QSG.* profiles defined
No QMGR.* profiles defined qsg-name.NO.QSG.CHECKS qmgr-name.YES.QSG.CHECKS
qsg-name.NO.QMGR.CHECKS qmgr-name.YES.QMGR.CHECKS qsg-name.NO.QSG.CHECKS qmgr-name.YES.QSG.CHECKS

Table 28. Valid security switch combinations for queue manager and queue sharing group level security (continued)

Combinations
No profiles for either switch defined

Table 29. Other valid security switch combinations that switch both levels of checking **on**.

Combinations
qmgr-name.NO.QMGR.CHECKS qmgr-name.NO.QSG.CHECKS
qsg-name.NO.QMGR.CHECKS qsg-name.NO.QSG.CHECKS
qmgr-name.NO.QMGR.CHECKS qsg-name.NO.QSG.CHECKS
qsg-name.NO.QMGR.CHECKS qmgr-name.NO.QSG.CHECKS

z/OS *Resource level checks*

A number of switch profiles are used to control access to resources. Some stop checking being performed on either a queue manager or a queue sharing group. These can be overridden by profiles that enable checking for specific queue managers.

Table 30 on page 192 shows the switch profiles used to control access to IBM MQ resources.

If your queue manager is part of a queue sharing group and you have both queue manager and queue sharing group security active, you can use a YES.* switch profile to override queue sharing group level profiles and specifically turn on security for a particular queue manager.

Some profiles apply to both queue managers and queue sharing groups. These are prefixed by the string *hlq* and you should substitute the name of your queue sharing group or queue manager, as applicable. Profile names shown prefixed by *qmgr-name* are queue manager override profiles; you should substitute the name of your queue manager.

Table 30. Switch profiles for resource checking

Type of resource checking that is controlled	Switch profile name	Override profile for a particular queue manager
Connection security	hlq.NO.CONNECT.CHECKS	qmgr-name.YES.CONNECT.CHECKS
Queue security	hlq.NO.QUEUE.CHECKS	qmgr-name.YES.QUEUE.CHECKS
Process security	hlq.NO.PROCESS.CHECKS	qmgr-name.YES.PROCESS.CHECKS
Namelist security	hlq.NO.NLIST.CHECKS	qmgr-name.YES.NLIST.CHECKS
Context security	hlq.NO.CONTEXT.CHECKS	qmgr-name.YES.CONTEXT.CHECKS
Alternate user security	hlq.NO.ALTERNATE.USER.CHECKS	qmgr-name.YES.ALTERNATE.USER.CHECKS
Command security	hlq.NO.CMD.CHECKS	qmgr-name.YES.CMD.CHECKS

Table 30. Switch profiles for resource checking (continued)

Type of resource checking that is controlled	Switch profile name	Override profile for a particular queue manager
Command resource security	hlq.NO.CMD.RESC.CHECKS	qmgr-name.YES.CMD.RESC.CHECKS
Topic security	hlq.NO.TOPIC.CHECKS	qmgr-name.YES.TOPIC.CHECKS
Note: Generic switch profiles such as hlq.NO.** are ignored by IBM MQ		

For example, if you want to perform process security checks on queue manager QM01, which is a member of queue sharing group QSG3 but you do not want to perform process security checks on any of the other queue managers in the group, define the following switch profiles:

```
QSG3.NO.PROCESS.CHECKS
QM01.YES.PROCESS.CHECKS
```

If you want to have queue security checks performed on all the queue managers in the queue sharing group, except QM02, define the following switch profile:

```
QM02.NO.QUEUE.CHECKS
```

(There is no need to define a profile for the queue sharing group because the checks are automatically enabled if there is no profile defined.)

z/OS *An example of defining switches*

Different IBM MQ subsystems have different security requirements, which can be implemented using different switch profiles.

Four IBM MQ subsystems have been defined:

- MQP1 (a production system)
- MQP2 (a production system)
- MQD1 (a development system)
- MQT1 (a test system)

All four queue managers are members of queue sharing group QS01. All IBM MQ RACF classes have been defined and activated.

These subsystems have different security requirements:

- The production systems require full IBM MQ security checking to be active at queue sharing group level on both systems.

  This is done by specifying the following profile:

  ```
  RDEFINE MQADMIN QS01.NO.QMGR.CHECKS
  ```

  This sets queue sharing group level checking for all the queue managers in the queue sharing group. You do not need to define any other switch profiles for the production queue managers because you want to check everything for these systems.
- Test queue manager MQT1 also requires full security checking. However, because you might want to change this later, security can be defined at queue manager level so that you can change the security settings for this queue manager without affecting the other members of the queue sharing group.

  This is done by defining the NO.QSG.CHECKS profile for MQT1 as follows:

```
RDEFINE MQADMIN MQT1.NO.QSG.CHECKS
```

- Development queue manager MQD1 has different security requirements from the rest of the queue sharing group. It requires only connection and queue security to be active.

  This is done by defining a `MQD1.YES.QMGR.CHECKS` profile for this queue manager, and then defining the following profiles to switch off security checking for the resources that do not need to be checked:

  ```
  RDEFINE MQADMIN MQD1.NO.CMD.CHECKS
  RDEFINE MQADMIN MQD1.NO.CMD.RESC.CHECKS
  RDEFINE MQADMIN MQD1.NO.PROCESS.CHECKS
  RDEFINE MQADMIN MQD1.NO.NLIST.CHECKS
  RDEFINE MQADMIN MQD1.NO.CONTEXT.CHECKS
  RDEFINE MQADMIN MQD1.NO.ALTERNATE.USER.CHECKS
  ```

When the queue manager is active, you can display the current security settings by issuing the DISPLAY SECURITY MQSC command.

You can also change the switch settings when the queue manager is running by defining or deleting the appropriate switch profile in the MQADMIN class. To make the changes to the switch settings active, you must issue the REFRESH SECURITY command for the MQADMIN class.

See "Refreshing queue manager security on z/OS" on page 245 for more details about using the DISPLAY SECURITY and REFRESH SECURITY commands.

z/OS Profiles used to control access to IBM MQ resources

You must define RACF profiles to control access to IBM MQ resources, in addition to the switch profiles that might have been defined. This collection of topics contains information about the RACF profiles for the different types of IBM MQ resource.

If you do not have a resource profile defined for a particular security check, and a user issues a request that would involve making that check, IBM MQ denies access. You do not have to define profiles for security types relating to any security switches that you have deactivated.

z/OS *Profiles for connection security*

If connection security is active, you must define profiles in the MQCONN class and permit the necessary groups or user IDs access to those profiles, so that they can connect to IBM MQ.

To enable a connection to be made, you must grant users RACF READ access to the appropriate profile. (If no queue manager level profile exists, and your queue manager is a member of a queue sharing group, checks might be made against queue sharing group level profiles, if the security is set up to do this.)

A connection profile qualified with a queue manager name controls access to a specific queue manager and users given access to this profile can connect to that queue manager. A connection profile qualified with queue sharing group name controls access to all queue managers within the queue sharing group for that connection type. For example, a user with access to `QS01.BATCH` can use a batch connection to any queue manager in queue sharing group QS01 that has not got a queue manager level profile defined.

Note:

1. For information about the user IDs checked for different security requests, see "User IDs for security checking on z/OS" on page 234.
2. Resource level security (RESLEVEL) checks are also made at connection time. For details, see "The RESLEVEL security profile" on page 228.

IBM MQ security recognizes the following different types of connection:

- Batch (and batch-type) connections, these include:
  - z/OS batch jobs
  - TSO applications

  - z/OS UNIX System Services sign-ons
  - Db2 stored procedures
- CICS connections
- IMS connections from control and application processing regions
- The IBM MQ channel initiator

z/OS *Connection security profiles for batch connections*
Profiles for checking batch-type connections are composed of the queue manager or queue sharing group name followed by the word *BATCH*. Give the user ID associated with the connecting address space READ access to the connection profile.

Profiles for checking batch and batch-type connections take the form:

```
hlq.BATCH
```

where `hlq` can be either the `qmgr-name` (queue manager name) or `qsg-name` (queue sharing group name). If you are using both queue manager and queue sharing group level security, IBM MQ checks for a profile prefixed by the queue manager name. If it does not find one, it looks for a profile prefixed by the queue sharing group name. If it fails to find either profile, the connection request fails.

For batch or batch-type connection requests, you must permit the user ID associated with the connecting address space to access the connection profile. For example, the following RACF command allows users in the CONNTQM1 group to connect to the queue manager TQM1; these user IDs will be permitted to use any batch or batch-type connection.

```
RDEFINE MQCONN TQM1.BATCH UACC(NONE)
PERMIT TQM1.BATCH CLASS(MQCONN) ID(CONNTQM1) ACCESS(READ)
```

z/OS *Using **CHCKLOCL** on locally bound applications*
CHCKLOCL only applies to connections that are made through BATCH connections and does not apply to connections made from CICS or IMS. Connections made through the channel initiator are controlled by **CHCKCLNT**.

Overview

If you want to configure your z/OS queue manager to mandate user ID and password checking for some, but not all, of your locally bound applications, you need to do some additional configuration.

The reason for this is that once **CHCKLOCL** (*REQUIRED*) is configured, legacy batch applications that use the MQCONN API call can no longer connect to the queue manager.

For z/OS only, a more granular mechanism based on the connection security of an address space can be used to downgrade the global CHCKLOCL(REQUIRED) configuration to CHCKLOCL(OPTIONAL) for specifically defined user IDs. The mechanism used, is described in the following text, together with an example.

In order to allow more granularity on **CHCKLOCL** (*REQUIRED*) than just EVERYONE, you modify **CHCKLOCL** in the same manner as you modify the access level of the user ID associated with the connecting address space to the `hlq.batch` connection profiles in the MQCONN class.

If the address space user ID only has READ access, which is the minimum you require to be able to connect at all, the **CHCKLOCL** configuration applies as written.

If the address space user ID has UPDATE access (or above) then the **CHCKLOCL** configuration operates in *OPTIONAL* mode. That is, you do not have to provide a user ID and password, but if you do, the user ID and password must be a valid pair.

Connection security already configured for your z/OS queue manager

If you have connection security configured for your z/OS queue manager and you want **CHCKLOCL** (*REQUIRED*) to apply to WAS locally bound applications, and no others, carry out the following steps:

1. Start with **CHCKLOCL** (*OPTIONAL*) as your configuration. This means that any user ID and passwords that are supplied are checked for validity, but not mandated.
2. List all the users that have access to the connection security profiles by issuing the command:

   ```
   RLIST MQCONN MQ23.BATCH AUTHUSER
   ```

   This command displays, for example:

   ```
   CLASS    NAME
   -----    ----
   MQCONN   MQ23.BATCH

   USER     ACCESS  ACCESS COUNT
   ----     ------  ------ -----
   JOHNDOE  READ    000009
   JDOE1    READ    000003
   WASUSER  READ    000000
   ```
3. For each user ID listed as having READ access, change the access to

   ```
   UPDATE:- PERMIT MQ23.BATCH CLASS(MQCONN) ID(JOHNDOE) ACCESS(UPDATE)
   ```
4. Update the IBM MQ configuration to **CHCKLOCL** (*REQUIRED*).

   The combination of UPDATE access to `MQ23.BATCH` and the current setting means that you are using **CHCKLOCL** (*OPTIONAL*).
5. Now, apply the **CHCKLOCL** (*REQUIRED*) behavior to one specific user ID, for example `WASUSER`, so that all the connections coming from that region must provide a user ID and password.

   Do this by reversing the change you made previously, by issuing the command:

   ```
   PERMIT MQ23.BATCH CLASS(MQCONN) ID(WASUSER) ACCESS(READ)
   ```

Connection security is not configured for your z/OS queue manager

In this situation, you must:

1. Create connection profiles for `hlq`.BATCH in the MQCONN class, by issuing the command:

   ```
   RDEFINE MQCONN MQ23.BATCH UACC(NONE)
   ```
2. Authorize all user IDs that create batch connections to the queue manager, so that they have UPDATE access to this profile. Doing this bypasses the **CHCKLOCL** (*REQUIRED*) requirement for the user ID and password at the time of connection.

   Do this by issuing the command:

   ```
   PERMIT MQ23.BATCH CLASS(MQCONN)ID(JOHNDOE) ACCESS(UPDATE)
   ```

   These include user IDs:

   a. Used for CSQUTIL, ISPF panels, and other locally bound tools.
   b. Associated with batch like connections to the queue manager. Consider for example, Advanced Message Security, IBM Integration Bus, Db2 stored procedures, z/OS UNIX System Services and TSO users, and Java applications

3. Delete the switch profile for the queue manager by issuing the command:

```
hlq.NO.CONNECT.CHECKS
```

4. Now, apply the **CHCKLOCL** (*REQUIRED*) behavior to one specific user ID, for example WASUSER, so that all the connections coming from that region must provide a user ID and password.

   Do this by reversing the change you made previously, by issuing the command:

```
PERMIT MQ23.BATCH CLASS(MQCONN) ID(WASUSER) ACCESS(READ)
```

z/OS *Connection security profiles for CICS connections*

Profiles for checking CICS connections are composed of the queue manager or queue sharing group name followed by the word *CICS* . Give the user ID associated with the CICS address space READ access to the connection profile.

Profiles for checking connections from CICS take the form:

```
hlq.CICS
```

where `hlq` can be either `qmgr-name` (queue manager name) or `qsg-name` (queue sharing group name). If you are using both queue manager and queue sharing group level security, IBM MQ checks for a profile prefixed by the queue manager name. If it does not find one, it looks for a profile prefixed by the queue sharing group name. If it fails to find either profile, the connection request fails

For connection requests by CICS, you need only permit the CICS address space user ID access to the connection profile.

For example, the following RACF commands allow the CICS address space user ID KCBCICS to connect to the queue manager TQM1:

```
RDEFINE MQCONN TQM1.CICS UACC(NONE)
PERMIT TQM1.CICS CLASS(MQCONN) ID(KCBCICS) ACCESS(READ)
```

z/OS *Connection security profiles for IMS connections*

Profiles for checking IMS connections are composed of the queue manager or queue sharing group name followed by the word *IMS* . Give the IMS control and dependent region user IDs READ access to the connection profile.

Profiles for checking connections from IMS take the form:

```
hlq.IMS
```

where `hlq` can be either `qmgr-name` (queue manager name) or `qsg-name` (queue sharing group name). If you are using both queue manager and queue sharing group level security, IBM MQ checks for a profile prefixed by the queue manager name. If it does not find one, it looks for a profile prefixed by the queue sharing group name. If it fails to find either profile, the connection request fails

For connection requests by IMS, permit access to the connection profile for the IMS control and dependent region user IDs.

For example, the following RACF commands allow:

- The IMS region user ID, IMSREG, to connect to the queue manager TQM1.

- Users in group BMPGRP to submit BMP jobs.

```
RDEFINE MQCONN TQM1.IMS UACC(NONE)
PERMIT TQM1.IMS CLASS(MQCONN) ID(IMSREG,BMPGRP) ACCESS(READ)
```

z/OS *Connection security profiles for the channel initiator*

Profiles for checking connections from the channel initiator are composed of the queue manager or queue sharing group name followed by the word *CHIN*. Give the user ID used by the channel initiator started task address space READ access to the connection profile.

Profiles for checking connections from the channel initiator take the form:

```
hlq.CHIN
```

where `hlq` can be either `qmgr-name` (queue manager name) or `qsg-name` (queue sharing group name). If you are using both queue manager and queue sharing group level security, IBM MQ checks for a profile prefixed by the queue manager name. If it does not find one, it looks for a profile prefixed by the queue sharing group name. If it fails to find either profile, the connection request fails

For connection requests by the channel initiator, define access to the connection profile for the user ID used by the channel initiator started task address space.

For example, the following RACF commands allow the channel initiator address space running with user ID DQCTRL to connect to the queue manager TQM1:

```
RDEFINE MQCONN TQM1.CHIN UACC(NONE)
PERMIT TQM1.CHIN CLASS(MQCONN) ID(DQCTRL) ACCESS(READ)
```

z/OS *Profiles for queue security*

If queue security is active, you must define profiles in the appropriate classes and permit the necessary groups or user IDs access to these profiles. Queue security profiles are named after the queue manager or queue sharing group, and the queue to be opened.

If queue security is active, you must:

- Define profiles in the **MQQUEUE** or **GMQQUEUE** classes if using uppercase profiles.
- Define profiles in the **MXQUEUE** or **GMXQUEUE** classes if using mixed case profiles.
- Permit the necessary groups or user IDs access to these profiles, so that they can issue IBM MQ API requests that use queues.

Profiles for queue security take the form:

```
hlq.queuename
```

where `hlq` can be either `qmgr-name` (queue manager name) or `qsg-name` (queue sharing group name), and `queuename` is the name of the queue being opened, as specified in the object descriptor on the `MQOPEN` or `MQPUT1` call.

A profile prefixed by the queue manager name controls access to a single queue on that queue manager. A profile prefixed by the queue sharing group name controls access to access to one or more queues with that queue name on all queue managers within the queue sharing group, or access to a shared queue by any queue manager within the group. This access can be overridden on an individual queue manager by defining a queue manager level profile for that queue on that queue manager.

If your queue manager is a member of a queue sharing group and you are using both queue manager and queue sharing group level security, IBM MQ checks for a profile prefixed by the queue manager name first. If it does not find one, it looks for a profile prefixed by the queue sharing group name.

If you are using shared queues, you are recommended to use queue sharing group level security.

For details of how queue security operates when the queue name is that of an alias or a model queue, see "Considerations for alias queues" on page 200 and "Considerations for model queues" on page 201 .

The RACF access required to open a queue depends on the MQOPEN or MQPUT1 options specified. If more than one of the MQOO_* and MQPMO_* options is coded, the queue security check is performed for the highest RACF authority required.

Table 31. Access levels for queue security using the MQOPEN or MQPUT1 calls

MQOPEN or MQPUT1 option	RACF access level required to hlq.queuename
MQOO_BROWSE	READ
MQOO_INQUIRE	READ
MQOO_BIND_*	UPDATE
MQOO_INPUT_*	UPDATE
MQOO_OUTPUT or MQPUT1	UPDATE
MQOO_PASS_ALL_CONTEXT MQPMO_PASS_ALL_CONTEXT	UPDATE
MQOO_PASS_IDENTITY_CONTEXT MQPMO_PASS_IDENTITY_CONTEXT	UPDATE
MQOO_SAVE_ALL_CONTEXT	UPDATE
MQOO_SET_IDENTITY_CONTEXT MQPMO_SET_IDENTITY_CONTEXT	UPDATE
MQOO_SET_ALL_CONTEXT MQPMO_SET_ALL_CONTEXT	UPDATE
MQOO_SET	ALTER

For example, on IBM MQ queue manager QM77, all user IDs in the RACF group PAYGRP are to be given access to get messages from or put messages to all queues with names beginning with 'PAY.'. You can do this using these RACF commands:

```
RDEFINE MQQUEUE QM77.PAY.** UACC(NONE)
PERMIT QM77.PAY.** CLASS(MQQUEUE) ID(PAYGRP) ACCESS(UPDATE)
```

Also, all user IDs in the PAYGRP group must have access to put messages on queues that do not follow the PAY naming convention. For example:

```
REQUEST_QUEUE_FOR_PAYROLL
SALARY.INCREASE.SERVER
REPLIES.FROM.SALARY.MODEL
```

You can do this by defining profiles for these queues in the GMQQUEUE class and giving access to that class as follows:

```
RDEFINE GMQQUEUE PAYROLL.EXTRAS UACC(NONE)
        ADDMEM(QM77.REQUEST_QUEUE_FOR_PAYROLL,
               QM77.SALARY.INCREASE.SERVER,
               QM77.REPLIES.FROM.SALARY.MODEL)
PERMIT PAYROLL.EXTRAS CLASS(GMQQUEUE) ID(PAYGRP) ACCESS(UPDATE)
```

Note:

1. If the RACF access level that an application has to a queue security profile is changed, the changes only take effect for any new object handles obtained (that is, new `MQOPEN` s) for that queue. Those handles already in existence at the time of the change retain their existing access to the queue. If an application is required to use its changed access level to the queue rather than its existing access level, it must close and reopen the queue for each object handle that requires the change.
2. In the example, the queue manager name `QM77` could also be the name of a queue sharing group.

Other types of security checks might also occur at the time the queue is opened depending on the open options specified and the types of security that are active. See also "Profiles for context security" on page 214 and "Profiles for alternate user security" on page 212. For a summary table showing the open options and the security authorization needed when queue, context, and alternate user security are all active, see Table 36 on page 206.

If you are using publish/subscribe you must consider the following. When an MQSUB request is processed a security check is performed to ensure that the user ID making the request has the required access to put messages to the target IBM MQ queue as well as the required access to subscribe to the IBM MQ topic.

Table 32. Access levels for queue security using the MQSUB call

MQSUB option	**RACF access level required to hlq.queuename**
MQSO_ALTER, MQSO_CREATE, and MQSO_RESUME	UPDATE

Note:

1. The `hlq.queuename` is the destination queue for publications. When this is a managed queue, you need access to the appropriate model queue to be used for the managed queue and the dynamic queue that are created.
2. You can use a technique like this for the destination queue you provide on an MQSUB API call if you want to distinguish between the users making the subscriptions, and the users retrieving the publications from the destination queue.

z/OS *Considerations for alias queues*

When you issue an `MQOPEN` or `MQPUT1` call for an alias queue, IBM MQ makes a resource check against the queue name specified in the object descriptor (MQOD) on the call. It does not check if the user is allowed access to the target queue name.

For example, an alias queue called PAYROLL.REQUEST resolves to a target queue of PAY.REQUEST. If queue security is active, you need only be authorized to access the queue PAYROLL.REQUEST. No check is made to see if you are authorized to access the queue PAY.REQUEST.

z/OS *Using alias queues to distinguish between MQGET and MQPUT requests*

The range of MQI calls available in one access level can cause a problem if you want to restrict access to a queue to allow only the **`MQPUT`** call or only the **`MQGET`** call. A queue can be protected by defining two aliases that resolve to that queue: one that enables applications to get messages from the queue, and one that enable applications to put messages on the queue.

The following text gives you an example of how you can define your queues to IBM MQ:

```
DEFINE QLOCAL(MUST_USE_ALIAS_TO_ACCESS) GET(ENABLED)
       PUT(ENABLED)

DEFINE QALIAS(USE_THIS_ONE_FOR_GETS) GET(ENABLED)
       PUT(DISABLED) TARGET(MUST_USE_ALIAS_TO_ACCESS)

DEFINE QALIAS(USE_THIS_ONE_FOR_PUTS) GET(DISABLED)
       PUT(ENABLED) TARGET(MUST_USE_ALIAS_TO_ACCESS)
```

You must also make the following RACF definitions:

```
RDEFINE MQQUEUE hlq.MUST_USE_ALIAS_TO_ACCESS UACC(NONE)
RDEFINE MQQUEUE hlq.USE_THIS_ONE_FOR_GETS UACC(NONE)
RDEFINE MQQUEUE hlq.USE_THIS_ONE_FOR_PUTS UACC(NONE)
```

Then you ensure that no users have access to the queue hlq.MUST_USE_ALIAS_TO_ACCESS, and give the appropriate users or groups access to the alias. You can do this using the following RACF commands:

```
PERMIT hlq.USE_THIS_ONE_FOR_GETS CLASS(MQQUEUE)
       ID(GETUSER,GETGRP) ACCESS(UPDATE)
PERMIT hlq.USE_THIS_ONE_FOR_PUTS CLASS(MQQUEUE)
       ID(PUTUSER,PUTGRP) ACCESS(UPDATE)
```

This means user ID GETUSER and user IDs in the group GETGRP are only allowed to get messages on MUST_USE_ALIAS_TO_ACCESS through the alias queue USE_THIS_ONE_FOR_GETS; and user ID PUTUSER and user IDs in the group PUTGRP are only allowed to put messages through the alias queue USE_THIS_ONE_FOR_PUTS.

Note:

1. If you want to use a technique like this, you must inform your application developers, so that they can design their programs appropriately.
2. You can use a technique like this for the destination queue you provide on an MQSUB API request if you want to distinguish between the users making the subscriptions and the users 'getting' the publications from the destination queue.

z/OS *Considerations for model queues*

To open a model queue, you must be able to open both the model queue itself and the dynamic queue to which it resolves. Define generic RACF profiles for dynamic queues, including dynamic queues used by IBM MQ utilities.

When you open a model queue, IBM MQ security makes two queue security checks:

1. Are you authorized to access the model queue?
2. Are you authorized to access the dynamic queue to which the model queue resolves?

If the dynamic queue name contains a trailing asterisk (*) character, this * is replaced by a character string generated by IBM MQ, to create a dynamic queue with a unique name. However, because the whole name, including this generated string, is used for checking authority, you should define generic profiles for these queues.

For example, an MQOPEN call uses a model queue name of CREDIT.CHECK.REPLY.MODEL and a dynamic queue name of CREDIT.REPLY.* on queue manager (or queue sharing group) MQSP.

To do this, you must issue the following RACF commands to define the necessary queue profiles:

```
RDEFINE MQQUEUE MQSP.CREDIT.CHECK.REPLY.MODEL
RDEFINE MQQUEUE MQSP.CREDIT.REPLY.**
```

You must also issue the corresponding RACF PERMIT commands to allow the user access to these profiles.

A typical dynamic queue name created by an MQOPEN is something like CREDIT.REPLY.A346EF00367849A0. The precise value of the last qualifier is unpredictable; this is why you should use generic profiles for such queue names.

A number of IBM MQ utilities put messages on dynamic queues. You should define profiles for the following dynamic queue names, and provide RACF UPDATE access to the relevant user IDs (see "User IDs for security checking on z/OS" on page 234 for the correct user IDs):

```
SYSTEM.CSQUTIL.*  (used by CSQUTIL)
SYSTEM.CSQOREXX.* (used by the operations and control panels)
SYSTEM.CSQXCMD.*  (used by the channel initiator when processing CSQINPX)
CSQ4SAMP.*        (used by the IBM MQ supplied samples)
```

You might also consider defining a profile to control use of the dynamic queue name used by default in the application programming copy members. The IBM MQ-supplied copybooks contain a default *DynamicQName*, which is CSQ.*. This enables an appropriate RACF profile to be established.

Note: Do not allow application programmers to specify a single * for the dynamic queue name. If you do, you must define an hlq.** profile in the MQQUEUE class, and you would have to give it wide-ranging access. This means that this profile could also be used for other non-dynamic queues that do not have a more specific RACF profile. Your users could, therefore, gain access to queues you do not want them to access.

z/OS *Close options on permanent dynamic queues*

If an application opens a permanent dynamic queue that was created by another application and then attempts to delete that queue with an MQCLOSE option, some extra security checks are applied when the attempt is made.

Table 33. Access levels for close options on permanent dynamic queues

MQCLOSE option	RACF access level required to hlq.queuename
MQCO_DELETE	ALTER
MQCO_DELETE_PURGE	ALTER

z/OS *Security and remote queues*

When a message is put on a remote queue, the queue security that is implemented by the local queue manager depends on how the remote queue is specified when it is opened.

The following rules are applied:

1. If the remote queue has been defined on the local queue manager through the IBM MQ DEFINE QREMOTE command, the queue that is checked is the name of the remote queue. For example, if a remote queue is defined on queue manager MQS1 as follows:

   ```
   DEFINE QREMOTE(BANK7.CREDIT.REFERENCE)
          RNAME(CREDIT.SCORING.REQUEST)
          RQMNAME(BNK7)
          XMITQ(BANK1.TO.BANK7)
   ```

   In this case, a profile for BANK7.CREDIT.REFERENCE must be defined in the MQQUEUE class.

2. If the *ObjectQMgrName* for the request does not resolve to the local queue manager, a security check is carried out against the resolved (remote) queue manager name except in the case of a cluster queue where the check is made against the cluster queue name.

For example, the transmission queue BANK1.TO.BANK7 is defined on queue manager MQS1. An MQPUT1 request is then issued on MQS1 specifying *ObjectName* as BANK1.INTERBANK.TRANSFERS and an *ObjectQMgrName* of BANK1.TO.BANK7. In this case, the user performing the request must have access to BANK1.TO.BANK7.

3. If you make an MQPUT request to a queue and specify *ObjectQMgrName* as the name of an alias of the local queue manager, only the queue name is checked for security, not that of the queue manager.

When the message gets to the remote queue manager it might be subject to additional security processing. For more information, see "Security for remote messaging" on page 100.

z/OS *Dead-letter queue security*

Special considerations apply to the dead-letter queue, because many users must be able to put messages on it, but access to retrieve messages must be tightly restricted. You can achieve this by applying different RACF authorities to the dead-letter queue and an alias queue.

Undelivered messages can be put on a special queue called the dead-letter queue. If you have sensitive data that could possibly end up on this queue, you must consider the security implications of this because you do not want unauthorized users to retrieve this data.

Each of the following must be allowed to put messages onto the dead-letter queue:

- Application programs.
- The channel initiator address space and any MCA user IDs. (If the RESLEVEL profile is not present, or is defined so that channel user IDs are checked, the channel user ID also needs authority to put messages on the dead-letter queue.)
- CKTI, the CICS-supplied CICS task initiator.
- CSQQTRMN, the IBM MQ-supplied IMS trigger monitor.

The only application that can retrieve messages from the dead-letter queue should be a 'special' application that processes these messages. However, a problem arises if you give applications RACF UPDATE authority to the dead-letter queue for MQPUT s because they can then automatically retrieve messages from the queue using MQGET calls. You cannot disable the dead-letter queue for get operations because, if you do, not even the 'special' applications could retrieve the messages.

One solution to this problem is set up a two-level access to the dead-letter queue. CKTI, message channel agent transactions or the channel initiator address space, and 'special' applications have direct access; other applications can only access the dead-letter queue through an alias queue. This alias is defined to allow applications to put messages on the dead-letter queue, but not to get messages from it.

This is how it might work:

1. Define the real dead-letter queue with attributes PUT(ENABLED) and GET(ENABLED), as shown in the sample thlqual.SCSQPROC(CSQ4INYG).
2. Give RACF UPDATE authority for the dead-letter queue to the following user IDs:
   - User IDs that the CKTI and the MCAs or channel initiator address space run under.
   - The user IDs associated with the 'special' dead-letter queue processing application.
3. Define an alias queue that resolves to the real dead-letter queue, but give the alias queue these attributes: PUT(ENABLED) and GET(DISABLED). Give the alias queue a name with the same stem as the dead-letter queue name but append the characters ".PUT" to this stem. For example, if the dead-letter queue name is hlq.DEAD.QUEUE, the alias queue name would be hlq.DEAD.QUEUE.PUT.
4. To put a message on the dead-letter queue, an application uses the alias queue. This is what your application must do:
   - Retrieve the name of the real dead-letter queue. To do this, it opens the queue manager object using MQOPEN and then issues an MQINQ to get the dead-letter queue name.
   - Build the name of the alias queue by appending the characters '.PUT' to this name, in this case, hlq.DEAD.QUEUE.PUT.
   - Open the alias queue, hlq.DEAD.QUEUE.PUT.

- Put the message on the real dead-letter queue by issuing an MQPUT against the alias queue.

5. Give the user ID associated with the application RACF UPDATE authority to the alias, but no access (authority NONE) to the real dead-letter queue. This means that:

   - The application can put messages onto the dead-letter queue using the alias queue.
   - The application cannot get messages from the dead-letter queue using the alias queue because the alias queue is disabled for get operations.

   The application cannot get any messages from the real dead-letter queue either because it does have the correct RACF authority.

Table 34 on page 204 summarizes the RACF authority required for the various participants in this solution.

Table 34. RACF authority to the dead-letter queue and its alias

Associated user IDs	**Real dead-letter queue (hlq.DEAD.QUEUE)**	**Alias dead-letter queue (hlq.DEAD.QUEUE.PUT)**
MCA or channel initiator address space and CKTI	UPDATE	NONE
'Special' application (for dead-letter queue processing)	UPDATE	NONE
User-written application user IDs	NONE	UPDATE

If you use this method, the application cannot determine the maximum message length (MAXMSGL) of the dead-letter queue. This is because the MAXMSGL attribute cannot be retrieved from an alias queue. Therefore, your application should assume that the maximum message length is 100 MB, the maximum size IBM MQ for z/OS supports. The real dead-letter queue should also be defined with a MAXMSGL attribute of 100 MB.

Note: User-written application programs do not normally use alternate user authority to put messages on the dead-letter queue. This reduces the number of user IDs that have access to the dead-letter queue.

z/OS *System queue security*

You must set up RACF access to allow certain user IDs access to particular system queues.

Many of the system queues are accessed by the ancillary parts of IBM MQ:

- The CSQUTIL utility
- The message security policy utility (CSQ0UTIL)
- The operations and control panels
- The channel initiator address space (including the Queued Pub/Sub Daemon)
- The mqweb server, used by the IBM MQ Console and REST API.

The user IDs under which these run must be given RACF access to these queues, as shown in Table 35 on page 204.

Table 35. Access required to the SYSTEM queues by IBM MQ

SYSTEM queue	**CSQUTIL**	**CSQ0UTIL**	**mqweb server**	**Operations and control panels**	**Channel initiator for distributed queuing**
SYSTEM.ADMIN.CHANNEL.EVENT	-	-	-	-	UPDATE
SYSTEM.ADMIN.COMMAND.QUEUE	-	-	UPDATE	-	-
SYSTEM.BROKER.ADMIN.STREAM	-	-	-	-	ALTER

Table 35. Access required to the SYSTEM queues by IBM MQ (continued)					
SYSTEM queue	**CSQUTIL**	**CSQ0UTIL**	**mqweb server**	**Operations and control panels**	**Channel initiator for distributed queuing**
SYSTEM.BROKER.CONTROL.QUEUE	-	-	-	-	ALTER
SYSTEM.BROKER.DEFAULT.STREAM	-	-	-	-	ALTER
SYSTEM.BROKER.INTER.BROKER.COMMUNICATIONS	-	-	-	-	UPDATE
SYSTEM.CHANNEL.INITQ	-	-	-	-	UPDATE
SYSTEM.CHANNEL.SYNCQ	-	-	-	-	UPDATE
SYSTEM.CLUSTER.COMMAND.QUEUE	-	-	-	-	ALTER
SYSTEM.CLUSTER.REPOSITORY.QUEUE	-	-	-	-	UPDATE
SYSTEM.CLUSTER.TRANSMIT.QUEUE	-	-	-	-	ALTER
SYSTEM.COMMAND.INPUT	UPDATE	-	-	UPDATE	UPDATE
SYSTEM.COMMAND.REPLY.*	-	-	-	-	UPDATE
SYSTEM.COMMAND.REPLY.MODEL	UPDATE	-	-	UPDATE	UPDATE
SYSTEM.CSQOREXX.*	-	-	-	UPDATE	-
SYSTEM.CSQUTIL.*	UPDATE	-	-	-	-
SYSTEM.CSQXCMD.*	-	-	-	-	UPDATE
SYSTEM.HIERARCHY.STATE	-	-	-	-	UPDATE
SYSTEM.INTER.QMGR.CONTROL	-	-	-	-	UPDATE
SYSTEM.INTER.QMGR.PUBS	-	-	-	-	UPDATE
SYSTEM.INTER.QMGR.FANREQ	-	-	-	-	UPDATE
SYSTEM.PROTECTION.ERROR.QUEUE	-	-	-	-	UPDATE
SYSTEM.PROTECTION.POLICY.QUEUE	-	UPDATE "1" on page 205	-	-	READ
SYSTEM.QSG.CHANNEL.SYNCQ	-	-	-	-	UPDATE
SYSTEM.QSG.TRANSMIT.QUEUE	-	-	-	-	UPDATE
SYSTEM.REST.REPLY.QUEUE	-	-	UPDATE	-	-
SYSTEM.BLUEMIX.REGISTRATION.QUEUE	-	-	-	-	UPDATE

Notes:

1. The Advanced Message Security address space user also requires READ access to this queue.

z/OS *API-resource security access quick reference*
A summary of the **MQOPEN**, **MQPUT1**, **MQSUB**, and **MQCLOSE** options and the access required by the different resource security types.

Table 36. MQOPEN, MQPUT1, MQSUB, and MQCLOSE options and the security authorization required. Callouts shown like this **(1)** refer to the notes following this table.

	Minimum RACF access level required			
RACF class:	**MXTOPIC**	**MQQUEUE or MXQUEUE (1)**	**MQADMIN or MXADMIN**	**MQADMIN or MXADMIN**
RACF profile:	**(15 or 16)**	**(2)**	**(3)**	**(4)**
MQOPEN option				
MQOO_INQUIRE		READ **(5)**	No check	No check
MQOO_BROWSE		READ	No check	No check
MQOO_INPUT_*		UPDATE	No check	No check
MQOO_SAVE_ALL_CONTEXT **(6)**		UPDATE	No check	No check
MQOO_OUTPUT (USAGE=NORMAL) **(7)**		UPDATE	No check	No check
MQOO_PASS_IDENTITY_CONTEXT **(8)**		UPDATE	READ	No check
MQOO_PASS_ALL_CONTEXT **(8) (9)**		UPDATE	READ	No check
MQOO_SET_IDENTITY_CONTEXT **(8) (9)**		UPDATE	UPDATE	No check
MQOO_SET_ALL_CONTEXT **(8) (10)**		UPDATE	CONTROL	No check
MQOO_OUTPUT (USAGE (XMITQ) **(11)**		UPDATE	CONTROL	No check
MQOO_OUTPUT (topic object)	UPDATE **(16)**			
MQOO_OUTPUT (alias queue to topic object)	UPDATE **(16)**	UPDATE		
MQOO_SET		ALTER	No check	No check
MQOO_ALTERNATE_USER_AUTHORITY		**(12)**	**(12)**	UPDATE
MQPUT1 option				
Put on a normal queue **(7)**		UPDATE	No check	No check
MQPMO_PASS_IDENTITY_CONTEXT		UPDATE	READ	No check
MQPMO_PASS_ALL_CONTEXT		UPDATE	READ	No check
MQPMO_SET_IDENTITY_CONTEXT		UPDATE	UPDATE	No check
MQPMO_SET_ALL_CONTEXT		UPDATE	CONTROL	No check
MQOO_OUTPUT Put on a transmission queue **(11)**		UPDATE	CONTROL	No check
MQOO_OUTPUT (topic object)	UPDATE **(16)**			
MQOO_OUTPUT (alias queue to topic object)	UPDATE **(16)**	UPDATE		
MQPMO_ALTERNATE_USER_AUTHORITY		**(13)**	**(13)**	UPDATE
MQCLOSE option				
MQCO_DELETE **(14)**		ALTER	No check	No check

Table 36. MQOPEN, MQPUT1, MQSUB, and MQCLOSE options and the security authorization required. Callouts shown like this **(1)** refer to the notes following this table. *(continued)*

	Minimum RACF access level required			
RACF class:	**MXTOPIC**	**MQQUEUE or MXQUEUE (1)**	**MQADMIN or MXADMIN**	**MQADMIN or MXADMIN**
RACF profile:	**(15 or 16)**	**(2)**	**(3)**	**(4)**
MQCO_DELETE_PURGE **(14)**		ALTER	No check	No check
MQCO_REMOVE_SUB	ALTER **(15)**			
MQSUB option				
MQSO_CREATE	ALTER **(15)**	**(17)**	**(18)**	
MQSO_ALTER	ALTER **(15)**	**(17)**	**(18)**	
MQSO_RESUME	READ **(15)**	**(17)**	No check	
MQSO_ALTERNATE_USER_AUTHORITY				UPDATE
MQSO_SET_IDENTITY_CONTEXT			**(18)**	

Note:

1. This option is not restricted to queues. Use the MQNLIST or MXNLIST class for namelists, and the MQPROC or MXPROC class for processes.
2. Use RACF profile: hlq.resourcename
3. Use RACF profile: hlq.CONTEXT.queuename
4. Use RACF profile: hlq.ALTERNATE.USER. `alternateuserid`

   `alternateuserid` is the user identifier that is specified in the *AlternateUserId* field of the object descriptor. Note that up to 12 characters of the *AlternateUserId* field are used for this check, unlike other checks where only the first 8 characters of a user identifier are used.
5. No check is made when opening the queue manager for inquiries.
6. MQOO_INPUT_* must be specified as well. This is valid for a local, model or alias queue.
7. This check is done for a local or model queue that has a **Usage** queue attribute of MQUS_NORMAL, and also for an alias or remote queue (that is defined to the connected queue manager.) If the queue is a remote queue that is opened specifying an *ObjectQMgrName* (not the name of the connected queue manager) explicitly, the check is carried out against the queue with the same name as *ObjectQMgrName* (which must be a local queue with a **Usage** queue attribute of MQUS_TRANSMISSION).
8. MQOO_OUTPUT must be specified as well.
9. MQOO_PASS_IDENTITY_CONTEXT is implied as well by this option.
10. MQOO_PASS_IDENTITY_CONTEXT, MQOO_PASS_ALL_CONTEXT and MQOO_SET_IDENTITY_CONTEXT are implied as well by this option.
11. This check is done for a local or model queue that has a **Usage** queue attribute of MQUS_TRANSMISSION, and is being opened directly for output. It does not apply if a remote queue is being opened.
12. At least one of MQOO_INQUIRE, MQOO_BROWSE, MQOO_INPUT_*, MQOO_OUTPUT or MQOO_SET must be specified as well. The check carried out is the same as that for the other options specified.
13. The check carried out is the same as that for the other options specified.
14. This applies only for permanent dynamic queues that have been opened directly, that is, not opened through a model queue. No security is required to delete a temporary dynamic queue.

15. Use RACF profile hlq.SUBSCRIBE.topicname.
16. Use RACF profile hlq.PUBLISH.topicname.
17. If on the MQSUB request you specified a destination queue for the publications to be sent to, then a security check is carried out against that queue to ensure that you have put authority to that queue.
18. If on the MQSUB request, with MQSO_CREATE or MQSO_ALTER options specified, you want to set any of the identity context fields in the MQSD structure, you also need to specify the MQSO_SET_IDENTITY_CONTEXT option and you also need the appropriate authority to the context profile for the destination queue.

z/OS *Profiles for topic security*

If topic security is active, you must define profiles in the appropriate classes and permit the necessary groups or user IDs access to those profiles.

The concept of topic security within a topic tree is described in Publish/subscribe security.

If topic security is active, you must perform the following actions:

- Define profiles in the **MXTOPIC** or **GMXTOPIC** classes.
- Permit the necessary groups or user IDs access to these profiles, so that they can issue IBM MQ API requests that use topics.

Profiles for topic security take the form:

```
hlq.SUBSCRIBE.topicname
hlq.PUBLISH.topicname
```

where

- `hlq` is either `qmgr-name` (queue manager name) or `qsg-name` (queue sharing group name).
- `topicname` is the name of the topic administration node in the topic tree, associated either with the topic being subscribed to through an MQSUB call, or being published to through an MQOPEN call.

A profile prefixed by the queue manager name controls access to a single topic on that queue manager. A profile prefixed by the queue sharing group name controls access to one or more topics with that topic name on all queue managers within the queue sharing group. This access can be overridden on an individual queue manager by defining a queue manager level profile for that topic on that queue manager.

If your queue manager is a member of a queue sharing group and you are using both queue manager and queue sharing group level security, IBM MQ checks for a profile prefixed by the queue manager name first. If it does not find one, it looks for a profile prefixed by the queue sharing group name.

Subscribe

To subscribe to a topic, you need access to both the topic you are trying to subscribe to, and the destination queue for the publications.

When you issue an MQSUB request, the following security checks take place:

- Whether you have the appropriate level of access to subscribe to that topic, and also that the destination queue (if specified) is opened for output
- Whether you have the appropriate level of access to that destination queue.

Table 37. Access level required for topic security to subscribe

MQSUB option	RACF access required to `hlq.SUBSCRIBE.topicname` profile in MXTOPIC class
MQSO_CREATE and MQSO_ALTER	ALTER
MQSO_RESUME	READ

Table 38. Additional authority required to subscribe using a non-managed destination queue	
MQSUB option	**RACF access required to `hlq.CONTEXT.queuename` profile in MQADMIN or MXADMIN class**
MQSO_CREATE, MQSO_ALTER, and MQSO_RESUME	UPDATE
	RACF access required to `hlq.queuename` profile in MQQUEUE or MXQUEUE class
MQSO_CREATE and MQSO_ALTER	UPDATE
	RACF access required to `hlq.ALTERNATE.USER.alternateuserid` profile in MQADMIN or MXADMIN class
MQSO_ALTERNATE_USER_AUTHORITY	UPDATE

Considerations for managed queues for subscriptions

A security check is carried out to see if you are allowed to subscribe to the topic. However, no security checks are carried out when the managed queue is created, or to determine if you have access to put messages to this destination queue.

You cannot close delete a managed queue.

The model queues used are: `SYSTEM.DURABLE.MODEL.QUEUE` and `SYSTEM.NDURABLE.MODEL.QUEUE`.

The managed queues created from these model queues are of the form `SYSTEM.MANAGED.DURABLE.A346EF00367849A0` and `SYSTEM.MANAGED.NDURABLE.A346EF0036785EA0` where the last qualifier is unpredictable.

Do not give any user access to these queues. The queues can be protected using generic profiles of the form `SYSTEM.MANAGED.DURABLE.*` and `SYSTEM.MANAGED.NDURABLE.*` with no authorities granted.

Messages can be retrieved from these queues using the handle returned on the MQSUB request.

If you explicitly issue an MQCLOSE call for a subscription with the `MQCO_REMOVE_SUB` option specified, and you did not create the subscription you are closing under this handle, a security check is performed at the time of closure to ensure that you have the correct authority to perform the operation.

Table 39. Access level required to profiles for topic security for closure of a subscribe operation	
MQCLOSE option	**RACF access required to `hlq.SUBSCRIBE.topicname` profile in MXTOPIC class**
MQCO_REMOVE_SUB	ALTER

Publish

To publish on a topic you need access to the topic and, if you are using alias queues, to the alias queue as well.

Table 40. Access level required for topic security to publish	
MQOPEN or MQPUT1 option	**RACF access required to `hlq.PUBLISH.topicname` profile in MXTOPIC class**
MQOO_OUTPUT or MQPUT1	UPDATE

Table 41. Access level required to open an alias queue that resolves to a topic

MQOPEN or MQPUT1 option	RACF access required to hlq.queuename profile in MQQUEUE or MXQUEUE class for the alias queue
MQOO_OUTPUT or MQPUT1	UPDATE

For details of how topic security operates when an alias queue that resolves to a topic name is opened for publish, see "Considerations for alias queues that resolve to topics for a publish operation" on page 210.

When you consider alias queues used for destination queues for PUT or GET restrictions, see "Considerations for alias queues" on page 200.

If the RACF access level that an application has to a topic security profile is changed, the changes take effect only for any new object handles obtained (that is, a new MQSUB or MQOPEN) for that topic. Those handles already in existence at the time of the change retain their existing access to the topic. Also, existing subscribers retain their access to any subscriptions that they have already made.

Considerations for alias queues that resolve to topics for a publish operation

When you issue an MQOPEN or MQPUT1 call for an alias queue that resolves to a topic, IBM MQ makes two resource checks:

- The first one against the alias queue name specified in the object descriptor (MQOD) on the MQOPEN or MQPUT1 call.
- The second against the topic to which the alias queue resolves

You must be aware that this behavior is different from the behavior you get when alias queues resolve to other queues. You need the correct access to both profiles in order for the publish action to proceed.

System topic security

The following system topics are accessed by the channel initiator address space.

The user IDs under which this runs must be given RACF access to these queues, as shown in Table 42 on page 210.

Table 42. Access required to the SYSTEM topics

SYSTEM topic	Profile	Channel initiator for distributed queuing
SYSTEM.BROKER.ADMIN.STREAM	hlq.PUBLISH.topicname	UPDATE
SYSTEM.BROKER.ADMIN.STREAM	hlq.SUBSCRIBE.topicname	ALTER

z/OS *Profiles for processes*

If process security is active, you must define profiles in the appropriate classes and permit the necessary groups or user IDs access to those profiles.

If process security is active, you must:

- Define profiles in the **MQPROC** or **GMQPROC** classes if using uppercase profiles.
- Define profiles in the **MXPROC** or **GMXPROC** classes if using mixed case profiles.
- Permit the necessary groups or user IDs access to these profiles, so that they can issue IBM MQ API requests that use processes.

Profiles for processes take the form:

```
hlq.processname
```

where `hlq` can be either `qmgr-name` (queue manager name) or `qsg-name` (queue sharing group name), and `processname` is the name of the process being opened.

A profile prefixed by the queue manager name controls access to a single process definition on that queue manager. A profile prefixed by the queue sharing group name controls access to one or more process definitions with that name on all queue managers within the queue sharing group. This access can be overridden on an individual queue manager by defining a queue manager level profile for that process definition on that queue manager.

If your queue manager is a member of a queue sharing group and you are using both queue manager and queue sharing group level security, IBM MQ checks for a profile prefixed by the queue manager name first. If it does not find one, it looks for a profile prefixed by the queue sharing group name.

The following table shows the access required for opening a process.

Table 43. Access levels for process security

MQOPEN option	RACF access level required to hlq.processname
MQOO_INQUIRE	READ

For example, on queue manager MQS9, the RACF group INQVPRC must be able to inquire (`MQINQ`) on all processes starting with the letter V. The RACF definitions for this would be:

```
RDEFINE MQPROC MQS9.V* UACC(NONE)
PERMIT MQS9.V* CLASS(MQPROC) ID(INQVPRC) ACCESS(READ)
```

Alternate user security might also be active, depending on the open options specified when a process definition object is opened.

z/OS *Profiles for namelists*

If namelist security is active, you define profiles in the appropriate classes and give the necessary groups or user IDs access to these profiles.

If namelist security is active, you must:

- Define profiles in the **`MQNLIST`** or **`GMQNLIST`** classes if using uppercase profiles.
- Define profiles in the **`MXNLIST`** or **`GMXNLIST`** classes if using mixed case profiles.
- Permit the necessary groups or user IDs access to these profiles.

Profiles for namelists take the form:

```
hlq.namelistname
```

where `hlq` can be either `qmgr-name` (queue manager name) or `qsg-name` (queue sharing group name), and `namelistname` is the name of the namelist being opened.

A profile prefixed by the queue manager name controls access to a single namelist on that queue manager. A profile prefixed by the queue sharing group name controls access to access to one or more namelists with that name on all queue managers within the queue sharing group. This access can be overridden on an individual queue manager by defining a queue manager level profile for that namelist on that queue manager.

If your queue manager is a member of a queue sharing group and you are using both queue manager and queue sharing group level security, IBM MQ checks for a profile prefixed by the queue manager name first. If it does not find one, it looks for a profile prefixed by the queue sharing group name.

The following table shows the access required for opening a namelist.

Table 44. Access levels for namelist security

MQOPEN option	RACF access level required to hlq.namelistname
MQOO_INQUIRE	READ

For example, on queue manager (or queue sharing group) PQM3, the RACF group DEPT571 must be able to inquire (`MQINQ`) on these namelists:

- All namelists starting with "DEPT571".
- PRINTER/DESTINATIONS/DEPT571
- AGENCY/REQUEST/QUEUES
- WAREHOUSE.BROADCAST

The RACF definitions to do this are:

```
RDEFINE MQNLIST PQM3.DEPT571.** UACC(NONE)
PERMIT PQM3.DEPT571.** CLASS(MQNLIST) ID(DEPT571) ACCESS(READ)

RDEFINE GMQNLIST NLISTS.FOR.DEPT571 UACC(NONE)
        ADDMEM(PQM3.PRINTER/DESTINATIONS/DEPT571,
               PQM3.AGENCY/REQUEST/QUEUES,
               PQM3.WAREHOUSE.BROADCAST)
PERMIT NLISTS.FOR.DEPT571 CLASS(GMQNLIST) ID(DEPT571) ACCESS(READ)
```

Alternate user security might be active, depending on the options specified when a namelist object is opened.

System namelist security

Many of the system namelists are accessed by the ancillary parts of IBM MQ:

- The CSQUTIL utility
- The operations and control panels
- The channel initiator address space (including the Queued Publish/Subscribe Daemon)

The user IDs under which these run must be given RACF access to these namelists, as shown in Table 45 on page 212.

Table 45. Access required to the SYSTEM namelists by IBM MQ

SYSTEM namelist	CSQUTIL	Operations and control panels	Channel initiator for distributed queuing
SYSTEM.QPUBSUB.QUEUE.NAMELIST	-	-	READ
SYSTEM.QPUBSUB.SUBPOINT.NAMELIST	-	-	READ

z/OS *Profiles for alternate user security*

If alternate user security is active, you must define profiles in the appropriate classes and permit the necessary groups or user IDs access to those profiles.

For more information about *AlternateUserId*, see AlternateUserID (MQCHAR12).

If alternate user security is active, you must:

- Define profiles in the MQADMIN or GMQADMIN classes if you are using uppercase profiles.
- Define profiles in the MXADMIN or GMXADMIN classes if you are using mixed case profiles.

Permit the necessary groups or user IDs access to these profiles, so that they can use the ALTERNATE_USER_AUTHORITY options when the object is opened.

Profiles for alternate user security can be specified at subsystem level or at queue sharing group level and take the following form:

```
hlq.ALTERNATE.USER.alternateuserid
```

Where `hlq` can be either `qmgr-name` (queue manager name) or `qsg-name` (queue sharing group name), and `alternateuserid` is the value of the *AlternateUserId* field in the object descriptor.

A profile prefixed by the queue manager name controls use of an alternative user ID on that queue manager. A profile prefixed by the queue sharing group name controls use of an alternative user ID on all queue managers within the queue sharing group. This alternative user ID can be used on any queue manager within the queue sharing group by a user that has the correct access. This access can be overridden on an individual queue manager by defining a queue manager level profile for that alternative user ID on that queue manager.

If your queue manager is a member of a queue sharing group and you are using both queue manager and queue sharing group level security, IBM MQ checks for a profile prefixed by the queue manager name first. If it does not find one, it looks for a profile prefixed by the queue sharing group name.

The following table shows the access when specifying an alternative user option.

Table 46. Access levels for alternate user security

MQOPEN, MQSUB, or MQPUT1 option	**RACF access level required**
MQOO_ALTERNATE_USER_AUTHORITY MQSO_ALTERNATE_USER_AUTHORITY MQPMO_ALTERNATE_USER_AUTHORITY	UPDATE

In addition to alternate user security checks, other security checks for queue, process, namelist, and context security can also be made. The alternative user ID, if provided, is only used for security checks on queue, process definition, or namelist resources. For alternate user and context security checks, the user ID requesting that the check is used. For details about how user IDs are handled, see "User IDs for security checking on z/OS" on page 234. For a summary table showing the open options and the security checks required when queue, context and alternate user security are all active, see Table 36 on page 206.

An alternative user profile gives the requesting user ID access to resources associated with the user ID specified in the alternative user ID. For example, the payroll server running under user ID PAYSERV on queue manager QMPY processes requests from personnel user IDs, all of which start with PS. To cause the work performed by the payroll server to be carried out under the user ID of the requesting user, alternative user authority is used. The payroll server knows which user ID to specify as the alternative user ID because the requesting programs generate messages using the MQPMO_DEFAULT_CONTEXT put message option. See "User IDs for security checking on z/OS" on page 234 for more details about from where alternative user IDs are obtained.

The following example RACF definitions enable the server program to specify alternative user IDs starting with the characters PS:

```
RDEFINE MQADMIN QMPY.ALTERNATE.USER.PS* UACC(NONE)
PERMIT QMPY.ALTERNATE.USER.PS* CLASS(MQADMIN) ID(PAYSERV) ACCESS(UPDATE)
```

Note:

1. The *AlternateUserId* fields in the object descriptor and subscription descriptor are 12 bytes long. All 12 bytes are used in the profile checks, but only the first 8 bytes are used as the user ID by IBM MQ. If this user ID truncation is not desirable, application programs making the request must translate any alternative user ID over 8 bytes into something more appropriate.
2. If you specify MQOO_ALTERNATE_USER_AUTHORITY, MQSO_ALTERNATE_USER_AUTHORITY, or MQPMO_ALTERNATE_USER_AUTHORITY and you do not specify an *AlternateUserId* field in the object descriptor, a user ID of blanks is used. For the purposes of the alternate user security check the user ID used for the *AlternateUserId* qualifier is -BLANK-. For example `RDEF MQADMIN hlq.ALTERNATE.USER.-BLANK-`.

   If the user is allowed to access this profile, all further checks are made with a user ID of blanks. For details of blank user IDs, see "Blank user IDs and UACC levels" on page 242.

The administration of alternative user IDs is easier if you have a naming convention for user IDs that enables you to use generic alternative user profiles. If they do not, you can use the RACF RACVAR feature. For details about using RACVAR, see the z/OS Security Server RACF documentation..

When a message is put to a queue that has been opened with alternative user authority and the context of the message has been generated by the queue manager, the MQMD_USER_IDENTIFIER field is set to the alternative user ID.

z/OS *Profiles for context security*

If context security is active, to control access to the message context information you must define profiles in the appropriate classes and permit the necessary groups or user IDs access to those profiles. The message context is contained within the message descriptor (MQMD).

Using profiles for context security

If context security is active, to permit users to access context information for messages on a particular queue, or when publishing to a particular topic, you must define a profile in one of the following classes:

- The `MQADMIN` class if using uppercase profiles.
- The `MXADMIN` class if using mixed-case profiles.

Profiles for context security can be specified at subsystem level or at queue sharing group level and take the following form:

```
hlq.CONTEXT.queuename
hlq.CONTEXT.topicname
```

where *hlq* can be either the queue manager name or the queue sharing group name, and *queuename* and *topicname* can be either the full or generic name of the queue or topic you want to define the context profile for.

A profile prefixed by the queue manager name, and with ** specified as the queue or topic name, allows control for context security on all queues and topics belonging to that queue manager. This can be overridden on an individual queue or topic by defining a specific profile for context on that queue or topic.

A profile prefixed by the queue sharing group name, and with ** specified as the queue or topic name, allows control for context on all queues and topics belonging to the queue managers within the queue sharing group. This can be overridden on an individual queue manager by defining a queue manager level profile for context on that queue manager, by specifying a profile prefixed by the queue manager name. It can also be overridden on an individual queue or topic by specifying a profile suffixed with the queue or topic name.

If your queue manager is a member of a queue sharing group and you are using both queue manager and queue sharing group level security, IBM MQ checks for a profile prefixed by the queue manager name first. If it does not find one, it looks for a profile prefixed by the queue sharing group name.

You must permit the necessary groups or user IDs access to this profile. The following table shows the access level required, depending on the specification of the context options when the queue is opened.

Table 47. Access levels for context security

MQOPEN or MQPUT1 option	**RACF access level required to hlq.CONTEXT.queuename or hlq.CONTEXT.topicname**
MQPMO_NO_CONTEXT	No context security check
MQPMO_DEFAULT_CONTEXT	No context security check
MQOO_SAVE_ALL_CONTEXT	No context security check
MQOO_PASS_IDENTITY_CONTEXT MQPMO_PASS_IDENTITY_CONTEXT	READ
MQOO_PASS_ALL_CONTEXT MQPMO_PASS_ALL_CONTEXT	READ
MQOO_SET_IDENTITY_CONTEXT MQPMO_SET_IDENTITY_CONTEXT	UPDATE
MQOO_SET_ALL_CONTEXT MQPMO_SET_ALL_CONTEXT	CONTROL
MQOO_OUTPUT or MQPUT1 (USAGE(XMITQ))	CONTROL
MQSUB option	
MQSO_SET_IDENTITY_CONTEXT **(Note 2)**	UPDATE

Note:

1. The user IDs used for distributed queuing require `CONTROL` access to `hlq.CONTEXT.queuename` to put messages on the destination queue. See "User IDs used by the channel initiator" on page 237 for information about the user IDs used.
2. If on the MQSUB request, with `MQSO_CREATE` or `MQSO_ALTER` options specified, you want to set any of the identity context fields in the MQSD structure, you need to specify the `MQSO_SET_IDENTITY_CONTEXT` option. You require also, the appropriate authority to the context profile for the destination queue.

If you put commands on the system-command input queue, use the default context put message option to associate the correct user ID with the command.

For example, the IBM MQ-supplied utility program CSQUTIL can be used to offload and reload messages in queues. When offloaded messages are restored to a queue, the CSQUTIL utility uses the `MQOO_SET_ALL_CONTEXT` option to return the messages to their original state. In addition to the queue security required by this open option, context authority is also required. For example, if this authority is required by the group BACKGRP on queue manager MQS1, this would be defined by:

```
RDEFINE MQADMIN MQS1.CONTEXT.** UACC(NONE)
PERMIT MQS1.CONTEXT.** CLASS(MQADMIN) ID(BACKGRP) ACCESS(CONTROL)
```

Depending on the options specified, and the types of security performed, other types of security checks might also occur when the queue is opened. These include queue security (see "Profiles for queue security" on page 198), and alternate user security (see "Profiles for alternate user security" on page 212). For a summary table showing the open options and the security checks required when queue, context and alternate user security are all active, see Table 36 on page 206.

System queue context security

Many of the system queues are accessed by the ancillary parts of IBM MQ, for example the channel initiator address space, and the mqweb server used by the IBM MQ Console and REST API.

The user IDs under which these run under must be given RACF access to these queues, as shown in Table 48 on page 216.

Table 48. Access required to the SYSTEM queues for context operations

SYSTEM queue	Channel initiator for distributed queuing	mqweb server
SYSTEM.ADMIN.COMMAND.QUEUE	-	CONTROL
SYSTEM.BROKER.CONTROL.QUEUE	CONTROL	-
SYSTEM.BROKER.INTER.BROKER.COMMUNICATIONS	CONTROL	-
SYSTEM.CHANNEL.SYNCQ	CONTROL	-
SYSTEM.CLUSTER.COMMAND.QUEUE	CONTROL	-
SYSTEM.CLUSTER.TRANSMIT.QUEUE	CONTROL	-

z/OS *Profiles for command security*

To enable security checking for commands, add profiles to the MQCMDS class. The profile names are based on the MQSC commands but control both MQSC and PCF commands. Profiles can apply to a queue manager or a queue sharing group.

If you want security checking for commands (so you have not defined the command security switch profile hlq.NO.CMD.CHECKS) you must add profiles to the MQCMDS class.

The same security profiles control both MQSC and PCF commands. The names of the RACF profiles for command security checking are based on the MQSC command names themselves. These profiles take the form:

```
hlq.verb.pkw
```

Where `hlq` can be either `qmgr-name` (queue manager name) or `qsg-name` (queue sharing group name), `verb` is the verb part of the command name, for example ALTER, and `pkw` is the object type, for example QLOCAL for a local queue.

Thus, the profile name for the ALTER QLOCAL command in subsystem CSQ1 is:

```
CSQ1.ALTER.QLOCAL
```

You can use generic profiles to protect sets of commands so that you have fewer profiles to maintain and, therefore, fewer access lists. Consider creating a generic profile that applies to all commands not protected by a more specific profile. Define this profile with UACC(NONE) and grant ALTER access only to the RACF groups containing administrators. You might then create a generic profile applicable to all DISPLAY commands and grant widespread access to it. Between these extremes, you might identify groups of users needing access to certain sets of commands, in which case you can create profiles for those sets and grant access to RACF groups representing those classes of user. Avoid giving users access to commands they do not require: Apply the principle of least privilege, so that users only have access to the commands that are required for their jobs.

A profile prefixed by the queue manager name controls the use of the command on that queue manager. A profile prefixed by the queue sharing group name controls the use of the command on all queue managers within the queue sharing group. This access can be overridden on an individual queue manager by defining a queue manager level profile for that command on that queue manager.

If your queue manager is a member of a queue sharing group and you are using both queue manager and queue sharing group level security, IBM MQ checks for a profile prefixed by the queue manager name. If it does not find one, it looks for a profile prefixed by the queue sharing group name.

By setting up command profiles at queue manager level, a user can be restricted from issuing commands on a particular queue manager. Alternatively, you can define one profile for a queue sharing group for each command verb, and all security checks take place against that profile instead of individual queue managers.

If both subsystem security and queue sharing group security are active and a local profile is not found, a command security check is performed to see if the user has access to a queue sharing group profile.

If you use the CMDSCOPE attribute to route a command to other queue managers in a queue sharing group, security is checked on each queue manager where the command is run, but not necessarily on the queue manager where the command is entered.

Table 49 on page 217 shows, for each IBM MQ MQSC command, the profiles required for command security checking to be carried out, and the corresponding access level for each profile in the MQCMDS class.

Table 50 on page 222 shows, for each IBM MQ PCF command, the profiles required for command security checking to be carried out, and the corresponding access level for each profile in the MQCMDS class.

Table 49. MQSC commands, profiles, and their access levels

Command	Command profile for MQCMDS	Access level for MQCMDS	Command resource profile for MQADMIN or MXADMIN	Access level for MQADMIN or MXADMIN
ALTER AUTHINFO	hlq.ALTER.AUTHINFO	ALTER	hlq.AUTHINFO.resourcename	ALTER
ALTER BUFFPOOL	hlq.ALTER.BUFFPOOL	ALTER	No check	-
ALTER CFSTRUCT	hlq.ALTER.CFSTRUCT	ALTER	No check	-
ALTER CHANNEL	hlq.ALTER.CHANNEL	ALTER	hlq.CHANNEL.channel	ALTER
ALTER NAMELIST	hlq.ALTER.NAMELIST	ALTER	hlq.NAMELIST.namelist	ALTER
ALTER PROCESS	hlq.ALTER.PROCESS	ALTER	hlq.PROCESS.process	ALTER
ALTER PSID	hlq.ALTER.PSID	ALTER	No check	-
ALTER QALIAS	hlq.ALTER.QALIAS	ALTER	hlq.QUEUE.queue	ALTER
ALTER QLOCAL“5” on page 222	hlq.ALTER.QLOCAL	ALTER	hlq.QUEUE.queue	ALTER
ALTER QMGR	hlq.ALTER.QMGR	ALTER	No check	-
ALTER QMODEL“5” on page 222	hlq.ALTER.QMODEL	ALTER	hlq.QUEUE.queue	ALTER
ALTER QREMOTE	hlq.ALTER.QREMOTE	ALTER	hlq.QUEUE.queue	ALTER
ALTER SECURITY	hlq.ALTER.SECURITY	ALTER	No check	-
ALTER SMDS	hlq.ALTER.SMDS	ALTER	No check	-
ALTER STGCLASS	hlq.ALTER.STGCLASS	ALTER	No check	-
ALTER SUB	hlq.ALTER.SUB	ALTER	No check	-
ALTER TOPIC	hlq.ALTER.TOPIC	ALTER	hlq.TOPIC.topic	ALTER
ALTER TRACE	hlq.ALTER.TRACE	ALTER	No check	-
ARCHIVE LOG	hlq.ARCHIVE.LOG	CONTROL	No check	-

Table 49. MQSC commands, profiles, and their access levels (continued)

Command	**Command profile for MQCMDS**	**Access level for MQCMDS**	**Command resource profile for MQADMIN or MXADMIN**	**Access level for MQADMIN or MXADMIN**
BACKUP CFSTRUCT	hlq.BACKUP.CFSTRUCT	CONTROL	No check	-
CLEAR QLOCAL	hlq.CLEAR.QLOCAL	ALTER	hlq.QUEUE.queue	ALTER
CLEAR TOPICSTR "3" on page 221	hlq.CLEAR.TOPICSTR	ALTER	hlq.TOPIC.topic	ALTER
DEFINE AUTHINFO	hlq.DEFINE.AUTHINFO	ALTER	hlq.AUTHINFO.resourcename	ALTER
DEFINE BUFFPOOL	hlq.DEFINE.BUFFPOOL	ALTER	No check	-
DEFINE CFSTRUCT	hlq.DEFINE.CFSTRUCT	ALTER	No check	-
DEFINE CHANNEL	hlq.DEFINE.CHANNEL	ALTER	hlq.CHANNEL.channel	ALTER
DEFINE LOG	hlq.DEFINE.LOG	ALTER	No check	-
DEFINE MAXSMSGS	hlq.DEFINE.MAXSMSGS	ALTER	No check	-
DEFINE NAMELIST	hlq.DEFINE.NAMELIST	ALTER	hlq.NAMELIST.namelist	ALTER
DEFINE PROCESS	hlq.DEFINE.PROCESS	ALTER	hlq.PROCESS.process	ALTER
DEFINE PSID	hlq.DEFINE.PSID	ALTER	No check	-
DEFINE QALIAS	hlq.DEFINE.QALIAS	ALTER	hlq.QUEUE.queue	ALTER
DEFINE QLOCAL"5" on page 222	hlq.DEFINE.QLOCAL	ALTER	hlq.QUEUE.queue	ALTER
DEFINE QMODEL"5" on page 222	hlq.DEFINE.QMODEL	ALTER	hlq.QUEUE.queue	ALTER
DEFINE QREMOTE	hlq.DEFINE.QREMOTE	ALTER	hlq.QUEUE.queue	ALTER
DEFINE STGCLASS	hlq.DEFINE.STGCLASS	ALTER	No check	-
DEFINE SUB	hlq.DEFINE.SUB	ALTER	No check	-
DEFINE TOPIC	hlq.DEFINE.TOPIC	ALTER	hlq.TOPIC.topic	ALTER
DELETE AUTHINFO	hlq.DELETE.AUTHINFO	ALTER	hlq.AUTHINFO.resourcename	ALTER
DELETE BUFFPOOL	hlq.DELETE.BUFFPOOL	ALTER	No check	-
DELETE CFSTRUCT	hlq.DELETE.CFSTRUCT	ALTER	No check	-
DELETE CHANNEL	hlq.DELETE.CHANNEL	ALTER	hlq.CHANNEL.channel	ALTER
DELETE NAMELIST	hlq.DELETE.NAMELIST	ALTER	hlq.NAMELIST.namelist	ALTER
DELETE PROCESS	hlq.DELETE.PROCESS	ALTER	hlq.PROCESS.process	ALTER
DELETE PSID	hlq.DELETE.PSID	ALTER	No check	-
DELETE QALIAS	hlq.DELETE.QALIAS	ALTER	hlq.QUEUE.queue	ALTER
DELETE QLOCAL	hlq.DELETE.QLOCAL	ALTER	hlq.QUEUE.queue	ALTER

Table 49. MQSC commands, profiles, and their access levels (continued)

Command	Command profile for MQCMDS	Access level for MQCMDS	Command resource profile for MQADMIN or MXADMIN	Access level for MQADMIN or MXADMIN
DELETE QMODEL	hlq.DELETE.QMODEL	ALTER	hlq.QUEUE.queue	ALTER
DELETE QREMOTE	hlq.DELETE.QREMOTE	ALTER	hlq.QUEUE.queue	ALTER
DELETE STGCLASS	hlq.DELETE.STGCLASS	ALTER	No check	-
DELETE SUB	hlq.DELETE.SUB	ALTER	No check	-
DELETE TOPIC	hlq.DELETE.TOPIC	ALTER	hlq.TOPIC.topic	ALTER
DISPLAY ARCHIVE "1" on page 221	hlq.DISPLAY.ARCHIVE	READ	No check	-
DISPLAY AUTHINFO	hlq.DISPLAY.AUTHINFO	READ	No check	-
DISPLAY CFSTATUS	hlq.DISPLAY.CFSTATUS	READ	No check	-
DISPLAY CFSTRUCT	hlq.DISPLAY.CFSTRUCT	READ	No check	-
DISPLAY CHANNEL	hlq.DISPLAY.CHANNEL	READ	No check	-
DISPLAY CHINIT	hlq.DISPLAY.CHINIT	READ	No check	-
DISPLAY CHLAUTH	hlq.DISPLAY.CHLAUTH	READ	No check	-
DISPLAY CHSTATUS	hlq.DISPLAY.CHSTATUS	READ	No check	-
DISPLAY CLUSQMGR	hlq.DISPLAY.CLUSQMGR	READ	No check	-
DISPLAY CMDSERV	hlq.DISPLAY.CMDSERV	READ	No check	-
DISPLAY CONN "1" on page 221	hlq.DISPLAY.CONN	READ	No check	-
DISPLAY GROUP	hlq.DISPLAY.GROUP	READ	No check	-
DISPLAY LOG "1" on page 221	hlq.DISPLAY.LOG	READ	No check	-
DISPLAY MAXSMSGS	hlq.DISPLAY.MAXSMSGS	READ	No check	-
DISPLAY NAMELIST	hlq.DISPLAY.NAMELIST	READ	No check	-
DISPLAY PROCESS	hlq.DISPLAY.PROCESS	READ	No check	-
DISPLAY PUBSUB	hlq.DISPLAY.PUBSUB	READ	No check	-
DISPLAY QALIAS	hlq.DISPLAY.QALIAS	READ	No check	-
DISPLAY QCLUSTER	hlq.DISPLAY.QCLUSTER	READ	No check	-
DISPLAY QLOCAL	hlq.DISPLAY.QLOCAL	READ	No check	-
DISPLAY QMGR	hlq.DISPLAY.QMGR	READ	No check	-
DISPLAY QMODEL	hlq.DISPLAY.QMODEL	READ	No check	-
DISPLAY QREMOTE	hlq.DISPLAY.QREMOTE	READ	No check	-
DISPLAY QSTATUS	hlq.DISPLAY.QSTATUS	READ	No check	-
DISPLAY QUEUE	hlq.DISPLAY.QUEUE	READ	No check	-

Table 49. MQSC commands, profiles, and their access levels (continued)

Command	Command profile for MQCMDS	Access level for MQCMDS	Command resource profile for MQADMIN or MXADMIN	Access level for MQADMIN or MXADMIN
DISPLAY SBSTATUS	hlq.DISPLAY.SBSTATUS	READ	No check	-
DISPLAY SMDS	hlq.DISPLAY.SMDS	READ	No check	-
DISPLAY SMDSCONN	hlq.DISPLAY.SMDSCONN	READ	No check	-
DISPLAY SUB	hlq.DISPLAY.SUB	READ	No check	-
DISPLAY SECURITY	hlq.DISPLAY.SECURITY	READ	No check	-
DISPLAY STGCLASS	hlq.DISPLAY.STGCLASS	READ	No check	-
DISPLAY SYSTEM "1" on page 221	hlq.DISPLAY.SYSTEM	READ	No check	-
DISPLAY THREAD	hlq.DISPLAY.THREAD	READ	No check	-
DISPLAY TPSTATUS	hlq.DISPLAY.TPSTATUS	READ	No check	-
DISPLAY TOPIC	hlq.DISPLAY.TOPIC	READ	No check	-
DISPLAY TPSTATUS	hlq.DISPLAY.TPSTATUS	READ	No check	-
DISPLAY TRACE	hlq.DISPLAY.TRACE	READ	No check	-
DISPLAY USAGE "1" on page 221	hlq.DISPLAY.USAGE	READ	No check	-
MOVE QLOCAL	hlq.MOVE.QLOCAL	ALTER	hlq.QUEUE.from-queue hlq.QUEUE.to-queue	ALTER
PING CHANNEL	hlq.PING.CHANNEL	CONTROL	hlq.CHANNEL.channel	CONTROL
RECOVER BSDS	hlq.RECOVER.BSDS	CONTROL	No check	-
RECOVER CFSTRUCT	hlq.RECOVER.CFSTRUCT	CONTROL	No check	-
REFRESH CLUSTER	hlq.REFRESH.CLUSTER	ALTER	No check	-
REFRESH QMGR	hlq.REFRESH.QMGR	ALTER	No check	-
REFRESH SECURITY	hlq.REFRESH.SECURITY	ALTER	No check	-
RESET CFSTRUCT	hlq.RESET.CFSTRUCT	CONTROL	No check	-
RESET CHANNEL	hlq.RESET.CHANNEL	CONTROL	hlq.CHANNEL.channel	CONTROL
RESET CLUSTER	hlq.RESET.CLUSTER	CONTROL	No check	-
RESET QMGR	hlq.RESET.QMGR	CONTROL	No check	-
RESET QSTATS	hlq.RESET.QSTATS	CONTROL	hlq.QUEUE.queue	CONTROL
RESET SMDS	hlq.RESET.SMDS	CONTROL	No check	-
RESET TPIPE	hlq.RESET.TPIPE	CONTROL	No check	-
RESOLVE CHANNEL	hlq.RESOLVE.CHANNEL	CONTROL	hlq.CHANNEL.channel	CONTROL
RESOLVE INDOUBT	hlq.RESOLVE.INDOUBT	CONTROL	No check	-
RESUME QMGR	hlq.RESUME.QMGR	CONTROL	No check	-

Table 49. MQSC commands, profiles, and their access levels (continued)				
Command	**Command profile for MQCMDS**	**Access level for MQCMDS**	**Command resource profile for MQADMIN or MXADMIN**	**Access level for MQADMIN or MXADMIN**
RVERIFY SECURITY	hlq.RVERIFY.SECURITY	ALTER	No check	-
SET ARCHIVE	hlq.SET.ARCHIVE	CONTROL	No check	-
SET CHLAUTH	hlq.SET.CHLAUTH	CONTROL	No check	-
SET LOG	hlq.SET.LOG	CONTROL	No check	-
SET SYSTEM	hlq.SET.SYSTEM	CONTROL	No check	-
START CHANNEL	hlq.START.CHANNEL	CONTROL	hlq.CHANNEL.channel	CONTROL
START CHINIT “4” on page 221	hlq.START.CHINIT	CONTROL	No check	-
START CMDSERV	hlq.START.CMDSERV	CONTROL	No check	-
START LISTENER	hlq.START.LISTENER	CONTROL	No check	-
START QMGR	None “2” on page 221	-	-	-
START SMDSCONN	hlq.START.SMDSCONN	CONTROL	No check	-
START TRACE	hlq.START.TRACE	CONTROL	No check	-
STOP CHANNEL	hlq.STOP.CHANNEL	CONTROL	hlq.CHANNEL.channel	CONTROL
STOP CHINIT	hlq.STOP.CHINIT	CONTROL	No check	-
STOP CMDSERV	hlq.STOP.CMDSERV	CONTROL	No check	-
STOP LISTENER	hlq.STOP.LISTENER	CONTROL	No check	-
STOP QMGR	hlq.STOP.QMGR	CONTROL	No check	-
STOP SMDSCONN	hlq.STOP.SMDSCONN	CONTROL	No check	-
STOP TRACE	hlq.STOP.TRACE	CONTROL	No check	-
SUSPEND QMGR	hlq.SUSPEND.QMGR	CONTROL	No check	-

Notes:

1. These commands might be issued internally by the queue manager; no authority is checked in these cases.
2. IBM MQ does not check the authority of the user who issues the START QMGR command. However, you can use RACF, or your alternative security facilities to control access to the START xxxxMSTR command that is issued as a result of the START QMGR command.

   This is done by controlling access to the MVS.START.STC.xxxxMSTR profile in the RACF operator commands (OPERCMDS) class. For details of this procedure, see Granting the user access to the RACF OPERCMDS class in *z/OS MVS Planning: Operations*. If you use this technique, and an unauthorized user tries to start the queue manager, it terminates with a reason code of 00F30216.
3. The **`hlq.TOPIC.topic`** resource refers to the Topic object derived from the TOPICSTR. For more details, see “Publish/subscribe security” on page 473
4. In IBM MQ for z/OS, the resource name MVS.START.STC.CSQ1CHIN has an additional JOBNAME qualifier appended. This can cause problems when starting the channel initiator.

To resolve the problem replace MVS.START.STC. *ssid* CHIN with a profile for a resource named MVS.START.STC. *ssid* CHIN **.*** or MVS.START.STC. *ssid* CHIN. *ssid* CHIN where *ssid* is the subsystem ID for the queue manager. This requires RACF UPDATE authority. For more details, see MVS™ Commands, RACF Access Authorities, and Resource Names in *z/OS MVS Planning: Operations*.

The START for *ssid* MSTR does not include the JOBNAME= parameter. For consistency, you might want to update the profile for MVS.START.STC.ssidMSTR to MVS.START.STC.ssidMSTR.*.

5. Setting the queue attribute STREAMQ to a non blank value also requires ALTER access level to MQADMIN or MXADMIN for `hlq.ALTER.streamQ`.

Table 50. PCF commands, profiles, and their access levels

Command	Command profile for MQCMDS	Access level for MQCMDS	Command resource profile for MQADMIN or MXADMIN	Access level for MQADMIN or MXADMIN
Backup CF Structure	hlq.BACKUP.CFSTRUCT	CONTROL	No check	-
Change Authentication Information Object	hlq.ALTER.AUTHINFO	ALTER	hlq.AUTHINFO.resourcename	ALTER
Change CF Structure	hlq.ALTER.CFSTRUCT	ALTER	No check	-
Change Channel	hlq.ALTER.CHANNEL	ALTER	hlq.CHANNEL.channel	ALTER
Change Namelist	hlq.ALTER.NAMELIST	ALTER	hlq.NAMELIST.namelist	ALTER
Change Process	hlq.ALTER.PROCESS	ALTER	hlq.PROCESS.process	ALTER
Change Queue"2" on page 225	hlq.ALTER.QUEUE	ALTER	hlq.QUEUE.queue	ALTER
Change Queue Manager	hlq.ALTER.QMGR	ALTER	No check	-
Change Security	hlq.ALTER.SECURITY	ALTER	No check	-
Change SMDS	hlq.ALTER.SMDS	ALTER	No check	-
Change Storage Class	hlq.ALTER.STGCLASS	ALTER	No check	-
Change Subscription	hlq.ALTER.SUB	ALTER	No check	-
Change Topic	hlq.ALTER.TOPIC	ALTER	hlq.TOPIC.topic	ALTER
Clear Queue	hlq.CLEAR.QLOCAL	ALTER	hlq.QUEUE.queue	ALTER
Clear Topic String "1" on page 225	hlq.CLEAR.TOPICSTR	ALTER	hlq.TOPIC.topic	ALTER
Copy Authentication Information Object	hlq.DEFINE.AUTHINFO	ALTER	hlq.AUTHINFO.resourcename	ALTER
Copy CF Structure	hlq.DEFINE.CFSTRUCT	ALTER	No check	-
Copy Channel	hlq.DEFINE.CHANNEL	ALTER	hlq.CHANNEL.channel	ALTER
Copy Namelist	hlq.DEFINE.NAMELIST	ALTER	hlq.NAMELIST.namelist	ALTER
Copy Process	hlq.DEFINE.PROCESS	ALTER	hlq.PROCESS.process	ALTER
Copy Queue	hlq.DEFINE.QUEUE	ALTER	hlq.QUEUE.queue	ALTER
Copy Subscription	hlq.DEFINE.SUB	ALTER	No check	-
Copy Storage Class	hlq.DEFINE.STGCLASS	ALTER	No check	-
Copy Topic	hlq.DEFINE.TOPIC	ALTER	hlq.TOPIC.topic	ALTER
Create Authentication Information Object	hlq.DEFINE.AUTHINFO	ALTER	hlq.AUTHINFO.resourcename	ALTER

Table 50. PCF commands, profiles, and their access levels (continued)

Command	Command profile for MQCMDS	Access level for MQCMDS	Command resource profile for MQADMIN or MXADMIN	Access level for MQADMIN or MXADMIN
Create CF Structure	hlq.DEFINE.CFSTRUCT	ALTER	No check	-
Create Channel	hlq.DEFINE.CHANNEL	ALTER	hlq.CHANNEL.channel	ALTER
Create Namelist	hlq.DEFINE.NAMELIST	ALTER	hlq.NAMELIST.namelist	ALTER
Create Process	hlq.DEFINE.PROCESS	ALTER	hlq.PROCESS.process	ALTER
Create Queue“2” on page 225	hlq.DEFINE.QUEUE	ALTER	hlq.QUEUE.queue	ALTER
Create Storage Class	hlq.DEFINE.STGCLASS	ALTER	No check	-
Create Subscription	hlq.DEFINE.SUB	ALTER	No check	-
Create Topic	hlq.DEFINE.TOPIC	ALTER	hlq.TOPIC.topic	ALTER
Delete Authentication Information Object	hlq.DELETE.AUTHINFO	ALTER	hlq.AUTHINFO.resourcename	ALTER
Delete CF Structure	hlq.DELETE.CFSTRUCT	ALTER	No check	-
Delete Channel	hlq.DELETE.CHANNEL	ALTER	hlq.CHANNEL.channel	ALTER
Delete Namelist	hlq.DELETE.NAMELIST	ALTER	hlq.NAMELIST.namelist	ALTER
Delete Process	hlq.DELETE.PROCESS	ALTER	hlq.PROCESS.process	ALTER
Delete Queue	hlq.DELETE.QUEUE	ALTER	hlq.QUEUE.queue	ALTER
Delete Storage Class	hlq.DELETE.STGCLASS	ALTER	No check	-
Delete Subscription	hlq.DELETE.SUB	ALTER	No check	-
Delete Topic	hlq.DELETE.TOPIC	ALTER	hlq.TOPIC.topic	ALTER
Inquire Archive	hlq.DISPLAY.ARCHIVE	READ	No check	-
Inquire Authentication Information Object	hlq.DISPLAY.AUTHINFO	READ	No check	-
Inquire Authentication Information Object Names	hlq.DISPLAY.AUTHINFO	READ	No check	-
Inquire CF Structure	hlq.DISPLAY.CFSTRUCT	READ	No check	-
Inquire CF Structure Names	hlq.DISPLAY.CFSTRUCT	READ	No check	-
Inquire CF Structure Status	hlq.DISPLAY.CFSTATUS	READ	No check	-
Inquire Channel	hlq.DISPLAY.CHANNEL	READ	No check	-
Inquire Channel Authentication Records	hlq.DISPLAY.CHLAUTH	READ	No check	-
Inquire Channel Initiator	hlq.DISPLAY.CHINIT	READ	No check	-
Inquire Channel Names	hlq.DISPLAY.CHANNEL	READ	No check	-
Inquire Channel Status	hlq.DISPLAY.CHSTATUS	READ	No check	-
Inquire Cluster Queue Manager	hlq.DISPLAY.CLUSQMGR	READ	No check	-

Table 50. PCF commands, profiles, and their access levels (continued)

Command	Command profile for MQCMDS	Access level for MQCMDS	Command resource profile for MQADMIN or MXADMIN	Access level for MQADMIN or MXADMIN
Inquire Connection	hlq.DISPLAY.CONNPCF	READ	No check	-
Inquire Group	hlq.DISPLAY.GROUP	READ	No check	-
Inquire Log	hlq.DISPLAY.LOG	READ	No check	-
Inquire Namelist	hlq.DISPLAY.NAMELIST	READ	No check	-
Inquire Namelist Names	hlq.DISPLAY.NAMELIST	READ	No check	-
Inquire Process	hlq.DISPLAY.PROCESS	READ	No check	-
Inquire Process Names	hlq.DISPLAY.PROCESS	READ	No check	-
Inquire Pub/Sub Status	hlq.DISPLAY.PUBSUB	READ	No check	-
Inquire Queue	hlq.DISPLAY.QUEUE	READ	No check	-
Inquire Queue Manager	hlq.DISPLAY.QMGR	READ	No check	-
Inquire Queue Names	hlq.DISPLAY.QUEUE	READ	No check	-
Inquire Queue Status	hlq.DISPLAY.QSTATUS	READ	No check	-
Inquire Security	hlq.DISPLAY.SECURITY	READ	No check	-
Inquire SMDS	hlq.DISPLAY.SMDS	READ	No check	-
Inquire SMDSCONN	hlq.DISPLAY.SMDSCONN	READ	No check	-
Inquire Storage Class	hlq.DISPLAY.STGCLASS	READ	No check	-
Inquire Storage Class Names	hlq.DISPLAY.STGCLASS	READ	No check	-
Inquire Subscription	hlq.INQUIRE.SUB	READ	No check	-
Inquire Subscription Status	hlq.INQUIRE.SBSTATUS	READ	No check	-
Inquire System	hlq.DISPLAY.SYSTEM	READ	No check	-
Inquire Topic	hlq.DISPLAY.TOPIC	READ	No check	-
Inquire Topic Names	hlq.DISPLAY.TOPIC	READ	No check	-
Inquire Topic Status	hlq.DISPLAY.TPSTATUS	READ	No check	-
Inquire Usage	hlq.DISPLAY.USAGE	READ	No check	-
Move Queue	hlq.MOVE.QLOCAL	ALTER	hlq.QUEUE.from-queue hlq.QUEUE.to-queue	ALTER
Ping Channel	hlq.PING.CHANNEL	CONTROL	hlq.CHANNEL.channel	CONTROL
Recover CF Structure	hlq.RECOVER.CFSTRUCT	CONTROL	No check	-
Refresh Cluster	hlq.REFRESH.CLUSTER	ALTER	No check	-
Refresh Queue Manager	hlq.REFRESH.QMGR	ALTER	No check	-
Refresh Security	hlq.REFRESH.SECURITY	ALTER	No check	-
Reset CF Structure	hlq.RESET.CFSTRUCT	CONTROL	No check	-
Reset Channel	hlq.RESET.CHANNEL	CONTROL	hlq.CHANNEL.channel	CONTROL

Table 50. PCF commands, profiles, and their access levels (continued)

Command	Command profile for MQCMDS	Access level for MQCMDS	Command resource profile for MQADMIN or MXADMIN	Access level for MQADMIN or MXADMIN
Reset Cluster	hlq.RESET.CLUSTER	CONTROL	No check	-
Reset Queue Manager	hlq.RESET.QMGR	CONTROL	No check	-
Reset Queue Statistics	hlq.RESET.QSTATS	CONTROL	hlq.QUEUE.queue	CONTROL
Reset SMDS	hlq.RESET.SMDS	CONTROL	No check	-
Resolve Channel	hlq.RESOLVE.CHANNEL	CONTROL	hlq.CHANNEL.channel	CONTROL
Resume Queue Manager	hlq.RESUME.QMGR	CONTROL	No check	-
Resume Queue Manager Cluster	hlq.RESUME.QMGR	CONTROL	No check	-
Reverify Security	hlq.RVERIFY.SECURITY	ALTER	No check	-
Set Archive	hlq.SET.ARCHIVE	CONTROL	No check	-
Set Channel Authentication Record	hlq.SET.CHLAUTH	CONTROL	No check	-
Set Log	hlq.SET.LOG	CONTROL	No check	-
Set System	hlq.SET.SYSTEM	CONTROL	No check	-
Start Channel	hlq.START.CHANNEL	CONTROL	hlq.CHANNEL.channel	CONTROL
Start Channel Initiator	hlq.START.CHINIT	CONTROL	No check	-
Start Channel Listener	hlq.START.LISTENER	CONTROL	No check	-
Start SMDS Connection	hlq.START.SMDSCONN	CONTROL	No check	-
Stop Channel	hlq.STOP.CHANNEL	CONTROL	hlq.CHANNEL.channel	CONTROL
Stop Channel Initiator	hlq.STOP.CHINIT	CONTROL	No check	-
Stop Channel Listener	hlq.STOP.LISTENER	CONTROL	No check	-
Stop SMDS Connection	hlq.STOP.SMDSCONN	CONTROL	No check	-
Suspend Queue Manager	hlq.SUSPEND.QMGR	CONTROL	No check	-
Suspend Queue Manager Cluster	hlq.SUSPEND.QMGR	CONTROL	No check	-

Notes:

1. The **`hlq.TOPIC.topic`** resource refers to the Topic object derived from the TOPICSTR. For more details, see "Publish/subscribe security" on page 473
2. Setting the queue attribute STREAMQ to a non blank value also requires ALTER access level to MQADMIN or MXADMIN for `hlq.ALTER.streamQ`.

See "IBM MQ Console - required command security profiles" on page 226 for details of the IBM MQ PCF profiles required, when using the IBM MQ Console.

z/OS *IBM MQ Console - required command security profiles*

Operations performed in the IBM MQ Console by a user in the `MQWebAdmin`, or `MQWebAdminRO`, role take place under the security context of the mqweb server started task user ID. If you want to use the IBM MQ Console, the mqweb server started task user ID needs authorization to issue certain PCF commands.

Table 51 on page 226 shows, for each IBM MQ PCF command, the command security profiles required, and the corresponding access level for each profile in the MQCMDS class needed by the IBM MQ Console.

Table 51. IBM MQ Console PCF commands, profiles, and their access levels

Command	**Command profile for MQCMDS**	**Access level for MQCMDS**	**Command resource profile for MQADMIN or MXADMIN**	**Access level for MQADMIN or MXADMIN**
Change Authentication Information Object	hlq.ALTER.AUTHINFO	ALTER	hlq.AUTHINFO.resourcename	ALTER
Change Channel	hlq.ALTER.CHANNEL	ALTER	hlq.CHANNEL.channel	ALTER
Change Queue	hlq.ALTER.QUEUE	ALTER	hlq.QUEUE.queue	ALTER
Change Queue Manager	hlq.ALTER.QMGR	ALTER	No check	-
Change Topic	hlq.ALTER.TOPIC	ALTER	hlq.TOPIC.topic	ALTER
Clear Queue	hlq.CLEAR.QLOCAL	ALTER	hlq.QUEUE.queue	ALTER
Create Authentication Information Object	hlq.DEFINE.AUTHINFO	ALTER	hlq.AUTHINFO.resourcename	ALTER
Create Channel	hlq.DEFINE.CHANNEL	ALTER	hlq.CHANNEL.channel	ALTER
Create Queue	hlq.DEFINE.QUEUE	ALTER	hlq.QUEUE.queue	ALTER
Create Subscription	hlq.DEFINE.SUB	ALTER	No check	-
Create Topic	hlq.DEFINE.TOPIC	ALTER	hlq.TOPIC.topic	ALTER
Delete Authentication Information Object	hlq.DELETE.AUTHINFO	ALTER	hlq.AUTHINFO.resourcename	ALTER
Delete Channel	hlq.DELETE.CHANNEL	ALTER	hlq.CHANNEL.channel	ALTER
Delete Queue	hlq.DELETE.QUEUE	ALTER	hlq.QUEUE.queue	ALTER
Delete Subscription	hlq.DELETE.SUB	ALTER	No check	-
Delete Topic	hlq.DELETE.TOPIC	ALTER	hlq.TOPIC.topic	ALTER
Inquire Authentication Information Object	hlq.DISPLAY.AUTHINFO	READ	No check	-
Inquire Authentication Information Object Names	hlq.DISPLAY.AUTHINFO	READ	No check	-
Inquire Channel	hlq.DISPLAY.CHANNEL	READ	No check	-
Inquire Channel Authentication Records	hlq.DISPLAY.CHLAUTH	READ	No check	-
Inquire Channel Initiator	hlq.DISPLAY.CHINIT	READ	No check	-
Inquire Channel Names	hlq.DISPLAY.CHANNEL	READ	No check	-
Inquire Channel Status	hlq.DISPLAY.CHSTATUS	READ	No check	-
Inquire Queue	hlq.DISPLAY.QUEUE	READ	No check	-
Inquire Queue Manager	hlq.DISPLAY.QMGR	READ	No check	-

Table 51. IBM MQ Console PCF commands, profiles, and their access levels (continued)

Command	Command profile for MQCMDS	Access level for MQCMDS	Command resource profile for MQADMIN or MXADMIN	Access level for MQADMIN or MXADMIN
Inquire Queue Names	hlq.DISPLAY.QUEUE	READ	No check	-
Inquire Queue Status	hlq.DISPLAY.QSTATUS	READ	No check	-
Inquire Subscription	hlq.INQUIRE.SUB	READ	No check	-
Inquire Subscription Status	hlq.INQUIRE.SBSTATUS	READ	No check	-
Inquire Topic	hlq.DISPLAY.TOPIC	READ	No check	-
Inquire Topic Names	hlq.DISPLAY.TOPIC	READ	No check	-
Inquire Topic Status	hlq.DISPLAY.TPSTATUS	READ	No check	-
Ping Channel	hlq.PING.CHANNEL	CONTROL	hlq.CHANNEL.channel	CONTROL
Refresh Cluster	hlq.REFRESH.CLUSTER	ALTER	No check	-
Refresh Security	hlq.REFRESH.SECURITY	ALTER	No check	-
Reset Channel	hlq.RESET.CHANNEL	CONTROL	hlq.CHANNEL.channel	CONTROL
Resolve Channel	hlq.RESOLVE.CHANNEL	CONTROL	hlq.CHANNEL.channel	CONTROL
Set Channel Authentication Record	hlq.SET.CHLAUTH	CONTROL	No check	-
Start Channel	hlq.START.CHANNEL	CONTROL	hlq.CHANNEL.channel	CONTROL
Stop Channel	hlq.STOP.CHANNEL	CONTROL	hlq.CHANNEL.channel	CONTROL

z/OS *Profiles for command resource security*

If you have not defined the command resource security switch profile, because you want security checking for resources associated with commands, you must add resource profiles for each resource to the appropriate class. The same security profiles control both MQSC and PCF commands.

If you have not defined the command resource security switch profile, `hlq.NO.CMD.RESC.CHECKS`, because you want security checking for resources associated with commands, you must:

- Add a resource profile in the **MQADMIN** class, if using uppercase profiles, for each resource.
- Add a resource profile in the **MXADMIN** class, if using mixed case profiles, for each resource.

The same security profiles control both MQSC and PCF commands.

Profiles for command resource security checking take the form:

```
hlq.type.resourcename
```

where `hlq` can be either `qmgr-name` (queue manager name) or `qsg-name` (queue sharing group name).

A profile prefixed by the queue manager name controls access to the resources associated with commands on that queue manager. A profile prefixed by the queue sharing group name controls access to the resources associated with commands on all queue managers within the queue sharing group. This access can be overridden on an individual queue manager by defining a queue manager level profile for that command resource on that queue manager.

If your queue manager is a member of a queue sharing group and you are using both queue manager and queue sharing group level security, IBM MQ checks for a profile prefixed by the queue manager name first. If it does not find one, it looks for a profile prefixed by the queue sharing group name.

For example, the RACF profile name for command resource security checking against the model queue CREDIT.WORTHY in subsystem CSQ1 is:

```
CSQ1.QUEUE.CREDIT.WORTHY
```

Because the profiles for all types of command resource are held in the MQADMIN class, the "type" part of the profile name is needed in the profile to distinguish between resources of different types that have the same name. The "type" part of the profile name can be CHANNEL, QUEUE, TOPIC, PROCESS, or NAMELIST. For example, a user might be authorized to define hlq.QUEUE.PAYROLL.ONE, but not authorized to define hlq.PROCESS.PAYROLL.ONE

If the resource type is a queue, and the profile is a queue sharing group level profile, it controls access to one or more local queues within the queue sharing group, or access to a single shared queue from any queue manager in the queue sharing group.

MQSC commands, profiles, and their access levels shows, for each IBM MQ MQSC command, the profiles required for command security checking to be carried out, and the corresponding access level for each profile in the MQCMDS class.

PCF commands, profiles, and their access levels shows, for each IBM MQ PCF command, the profiles required for command security checking to be carried out, and the corresponding access level for each profile in the MQCMDS class.

z/OS *Command resource security checking for alias queues and remote queues*
Alias queue and remote queues both provide indirection to another queue. Additional points apply when you consider security checking for these queues.

Alias queues

When you define an alias queue, command resource security checks are only performed against the name of the alias queue, not against the name of the target queue to which the alias resolves.

Alias queues can resolve to both local and remote queues. If you do not want to permit users access to certain local or remote queues, you must do both of the following:

1. Do not allow the users access to these local and remote queues.
2. Restrict the users from being able to define aliases for these queues. That is, prevent them from being able to issue DEFINE QALIAS and ALTER QALIAS commands.

Remote queues

When you define a remote queue, command resource security checks are performed only against the name of the remote queue. No checks are performed against the names of the queues specified in the RNAME or XMITQ attributes in the remote queue object definition.

z/OS The RESLEVEL security profile

You can define a special profile in the MQADMIN or MXADMIN class to control the number of user IDs checked for API-resource security. This profile is called the RESLEVEL profile. How this profile affects API-resource security depends on how you access IBM MQ.

When an application tries to connect to IBM MQ, IBM MQ checks the access that the user ID associated with the connection has to a profile in the MQADMIN or MXADMIN class called:

```
hlq.RESLEVEL
```

Where `hlq` can be either `ssid` (subsystem ID) or `qsg` (queue sharing group ID).

The user IDs associated with each connection type are:

- The user ID of the connecting task for batch connections
- The CICS address space user ID for CICS connections
- The IMS region address space user ID for IMS connections
- The channel initiator address space user ID for channel initiator connections

Attention: RESLEVEL is a very powerful option; it can cause the bypassing of all resource security checks for a particular connection.

If you do not have a RESLEVEL profile defined, you must be careful that no other profile in the MQADMIN class matches hlq.RESLEVEL. For example, if you have a profile in MQADMIN called hlq.** and no hlq.RESLEVEL profile, beware of the consequences of the hlq.** profile because it is used for the RESLEVEL check.

Define an hlq.RESLEVEL profile and set the UACC to NONE, rather than have no RESLEVEL profile at all. Have as few users or groups in the access list as possible. For details about how to audit RESLEVEL access, see "Auditing considerations on z/OS" on page 253.

If you are using queue manager level security only, IBM MQ performs RESLEVEL checks against the `qmgr-name.`RESLEVEL profile. If you are using queue sharing group level security only, IBM MQ performs RESLEVEL checks against the `qsg-name.`RESLEVEL profile. If you are using a combination of both queue manager and queue sharing group level security, IBM MQ first checks for the existence of a RESLEVEL profile at queue manager level. If it does not find one, it checks for a RESLEVEL profile at queue sharing group level.

If it cannot find a RESLEVEL profile, IBM MQ enables checking of both the job and task (or alternate user) ID for a CICS or an IMS connection. For a batch connection, IBM MQ enables checking of the job (or alternate) user ID. For the channel initiator, IBM MQ enables checking of the channel user ID and the MCA (or alternate) user ID.

If there is a RESLEVEL profile, the level of checking depends on the environment and access level for the profile.

Remember that if your queue manager is a member of a queue sharing group and you do not define this profile at queue manager level, there might be one defined at queue sharing group level that will affect the level of checking. To activate the checking of two user IDs, you define a RESLEVEL profile (prefixed with either the queue manager name of the queue sharing group name) with a UACC(NONE) and ensure that the relevant users do not have access granted against this profile.

When you consider the access that the channel initiator's user ID has to RESLEVEL, remember that the connection established by the channel initiator is also the connection used by the channels. A setting that causes the bypassing of all resource security checks for the channel initiator's user ID effectively bypasses security checks for all channels. If the channel initiator's user ID access to RESLEVEL is something other than NONE, then only one user ID (for an access level of READ or UPDATE) or no user IDs (for an access level of CONTROL or ALTER) is checked for access. If you grant the channel initiator's user ID an access level other than NONE to RESLEVEL, be sure that you understand the effect of this setting on the security checks done for channels.

Using the RESLEVEL profile means that normal security audit records are not taken. For example, if you put UAUDIT on a user, the access to the hlq.RESLEVEL profile in MQADMIN is not audited.

If you use the RACF WARNING option on the hlq.RESLEVEL profile, no RACF warning messages are produced for profiles in the RESLEVEL class.

Security checking for report messages such as CODs are controlled by the RESLEVEL profile associated with the originating application. For example, if a batch job's userid has CONTROL or ALTER authority to a RESLEVEL profile, then all resource checking performed by the batch job are bypassed, including the security check of report messages.

If you change the RESLEVEL profile, users must disconnect and connect again before the change takes place. (This includes stopping and restarting the channel initiator if the access that the distributed queuing address space user ID has to the RESLEVEL profile is changed.)

To switch RESLEVEL auditing off, use the RESAUDIT system parameter.

z/OS RESLEVEL and batch connections

By default, when an IBM MQ resource is being accessed through batch and batch-type connections, the user must be authorized to access that resource for the particular operation. You can bypass the security check by setting up an appropriate RESLEVEL definition.

Whether the user is checked or not is based on the user ID used at connect time, the same user ID used for the connection check.

For example, you can set up RESLEVEL so that when a user you trust accesses certain resources through a batch connection, no API-resource security checks are done; but when a user you do not trust tries to access the same resources, security checks are carried out as normal. You should set up RESLEVEL checking to bypass API-resource security checks only when you sufficiently trust the user and the programs run by that user.

The following table shows the checks made for batch connections.

Table 52. Checks made at different RACF access levels for batch connections

RACF access level	Level of checking
NONE	Resource checks performed
READ	Resource checks performed
UPDATE	Resource checks performed
CONTROL	No check.
ALTER	No check.

z/OS RESLEVEL and system functions

The application of RESLEVEL to the operation and control panels, and to CSQUTIL.

The operation and control panels and the CSQUTIL utility are batch-type applications that make requests to the queue manager's command server, and so they are subject to the considerations described in "RESLEVEL and batch connections" on page 230. You can use RESLEVEL to bypass security checking for the SYSTEM.COMMAND.INPUT and SYSTEM.COMMAND.REPLY.MODEL queues that they use, but not for the dynamic queues SYSTEM.CSQXCMD.*, SYSTEM.CSQOREXX.*, and SYSTEM.CSQUTIL.*.

The command server is an integral part of the queue manager and so does not have connection or RESLEVEL checking associated with it. To maintain security, therefore, the command server must confirm that the user ID of the requesting application has authority to open the queue being used for replies. For the operations and control panels, this is SYSTEM.CSQOREXX.*. For CSQUTIL, it is SYSTEM.CSQUTIL.*. Users must be authorized to use these queues, as described in "System queue security" on page 204, in addition to any RESLEVEL authorization they are given.

For other applications using the command server, it is the queue they name as their reply-to queue. Such other applications might deceive the command server into placing messages on unauthorized queues by passing (in the message context) a more trusted user ID than its own to the command server. To prevent this, use a CONTEXT profile to protect the identity context of messages placed on SYSTEM.COMMAND.INPUT.

z/OS RESLEVEL and CICS connections

By default, when an API-resource security check is made on a CICS connection, two user IDs are checked. You can change which user IDs are checked by setting up a RESLEVEL profile.

The first user ID checked is that of the CICS address space. This is the user ID on the job card of the CICS job, or the user ID assigned to the CICS started task by the z/OS STARTED class or the started procedures table. (It is not the CICS DFLTUSER.)

The second user ID checked is the user ID associated with the CICS transaction.

If one of these user IDs does not have access to the resource, the request fails with a completion code of MQRC_NOT_AUTHORIZED. Both the CICS address space user ID and the user ID of the person running the CICS transaction must have access to the resource at the correct level.

How RESLEVEL can affect the checks made

Depending on how you set up your RESLEVEL profile, you can change which user IDs are checked when access to a resource is requested. See Table 53 on page 231 for more information.

The user IDs checked depend on the user ID used at connection time, that is, the CICS address space user ID. This control enables you to bypass API-resource security checking for IBM MQ requests coming from one system (for example, a test system, TESTCICS,) but to implement them for another (for example, a production system, PRODCICS).

Note: If you set up your CICS address space user ID with the "trusted" attribute in the STARTED class or the RACF started procedures table ICHRIN03, this overrides any user ID checks for the CICS address space established by the RESLEVEL profile for your queue manager (that is, the queue manager does not perform the security checks for the CICS address space). For more information, see Securing CICS.

The following table shows the checks made for CICS connections.

Table 53. Checks made at different RACF access levels for CICS connections

RACF access level	Level of checking
NONE	IBM MQ checks the CICS address space user ID and the transaction user ID.
READ	IBM MQ checks the CICS address space user ID only.
UPDATE	If the transaction is defined to CICS with RESSEC(YES), IBM MQ checks the CICS address space user ID and the transaction user ID.
UPDATE	If the transaction is defined to CICS with RESSEC(NO), IBM MQ checks the CICS address space user ID only.
CONTROL or ALTER	IBM MQ does not check any user IDs.

z/OS *RESLEVEL and IMS connections*

By default, when an API-resource security check is made for an IMS connection, two user IDs are checked. You can change which user IDs are checked by setting up a RESLEVEL profile.

By default, when an API-resource security check is made for an IMS connection, two user IDs are checked to see if access is allowed to the resource.

The first user ID checked is that of the address space of the IMS region. This is taken from either the USER field from the job card or the user ID assigned to the region from the z/OS STARTED class or the started procedures table (SPT).

The second user ID checked is associated with the work being done in the dependent region. It is determined according to the type of the dependent region as shown in How the second user ID is determined for the IMS(tm) connection.

If either the first or second IMS user ID does not have access to the resource, the request fails with a completion code of MQRC_NOT_AUTHORIZED.

The setting of IBM MQ RESLEVEL profiles cannot alter the user ID under which IMS transactions are scheduled from the IBM-supplied MQ-IMS trigger monitor program CSQQTRMN. This user ID is the PSBNAME of that trigger monitor, which by default is CSQQTRMN.

How RESLEVEL can affect the checks made

Depending on how you set up your RESLEVEL profile, you can change which user IDs are checked when access to a resource is requested. The possible checks are:

- Check the IMS region address space user ID and the second user ID or alternate user ID.
- Check IMS region address space user ID only.
- Do not check any user IDs.

The following table shows the checks made for IMS connections.

Table 54. Checks made at different RACF access levels for IMS connections

RACF access level	**Level of checking**
NONE	Check the IMS address space user ID and the IMS second user ID or alternate user ID.
READ	Check the IMS address space user ID.
UPDATE	Check the IMS address space user ID.
CONTROL	No check.
ALTER	No check.

z/OS *RESLEVEL and the channel initiator connection*

By default, when an API-resource security check is made by the channel initiator, two user IDs are checked. You can change which user IDs are checked by setting up a RESLEVEL profile.

By default, when an API-resource security check is made by the channel initiator, two user IDs are checked to see if access is allowed to the resource.

The user IDs checked can be that specified by the MCAUSER channel attribute, that received from the network, that of the channel initiator address space, or the alternate user ID for the message descriptor. Which user IDs are checked depends on the communication protocol you are using and the setting of the PUTAUT channel attribute. See "User IDs used by the channel initiator" on page 237 for more information.

If one of these user IDs does not have access to the resource, the request fails with a completion code of MQRC_NOT_AUTHORIZED.

How RESLEVEL can affect the checks made

Depending on how you set up your RESLEVEL profile, you can change which user IDs are checked when access to a resource is requested, and how many are checked.

The following table shows the checks made for the channel initiator's connection, and for all channels since they use this connection.

Table 55. Checks made at different RACF access levels for channel initiator connections

RACF access level	**Level of checking**
NONE	Check two user IDs.
READ	Check one user ID.
UPDATE	Check one user ID.
CONTROL	No check.
ALTER	No check.
Note: See "User IDs used by the channel initiator" on page 237 for a definition of the user IDs checked	

z/OS RESLEVEL and intra-group queuing

By default, when an API-resource security check is made by the intra-group queuing agent, two user IDs are checked to see if access is allowed to the resource. You can change which user IDs are checked by setting up an RESLEVEL profile.

The user IDs checked can be the user ID determined by the IGQUSER attribute of the receiving queue manager, the user ID of the queue manager within the queue sharing group that put the message on to the SYSTEM.QSG.TRANSMIT.QUEUE, or the alternate user ID specified in the `UserIdentifier` field of the message descriptor of the message. See "User IDs used by the intra-group queuing agent" on page 241 for more information.

Because the intra-group queuing agent is an internal queue manager task, it does not issue an explicit connect request and runs under the user ID of the queue manager. The intra-group queuing agent starts at queue manager initialization. During the initialization of the intra-group queuing agent, IBM MQ checks the access that the user ID associated with the queue manager has to a profile in the MQADMIN class called:

```
hlq.RESLEVEL
```

This check is always performed unless the hlq.NO.SUBSYS.SECURITY switch has been set.

If there is no RESLEVEL profile, IBM MQ enables checking for two user IDs. If there is a RESLEVEL profile, the level of checking depends on the access level granted to the user ID of the queue manager for the profile. Checks made at different RACF(r) access levels for the intra-group queuing agent shows the checks made for the intra-group queuing agent.

Table 56. Checks made at different RACF access levels for the intra-group queuing agent

RACF access level	Level of checking
NONE	Check two user IDs.
READ	Check one user ID.
UPDATE	Check one user ID.
CONTROL	No check.
ALTER	No check.

Note: See "User IDs used by the intra-group queuing agent" on page 241 for a definition of the user IDs checked

If the permissions granted to the RESLEVEL profile for the queue manager's user ID are changed, the intra-group queuing agent must be stopped and restarted to pick up the new permissions. Because there is no way to independently stop and restart the intra-group queuing agent, the queue manager must be stopped and restarted to achieve this.

z/OS RESLEVEL and the user IDs checked

Example of setting a RESLEVEL profile and granting access to it.

User ID checking against profile name for batch connections through User IDs checked against profile name for LU 6.2 and TCP/IP server-connection channels show how RESLEVEL affects which user IDs are checked for different MQI requests.

For example, you have a queue manager called QM66 with the following requirements:

- User WS21B is to be exempt from resource security.
- CICS started task WXNCICS running under address space user ID CICSWXN is to perform full resource checking only for transactions defined with RESSEC(YES).

To define the appropriate RESLEVEL profile, issue the following RACF command:

```
RDEFINE MQADMIN QM66.RESLEVEL UACC(NONE)
```

Then give the users access to this profile, using the following commands:

```
PERMIT QM66.RESLEVEL CLASS(MQADMIN) ID(WS21B) ACCESS(CONTROL)
PERMIT QM66.RESLEVEL CLASS(MQADMIN) ID(CICSWXN) ACCESS(UPDATE)
```

If you make these changes while the user IDs are connected to queue manager QM66, the users must disconnect and connect again before the change takes place.

If subsystem security is not active when a user connects but, while this user is still connected, subsystem security becomes active, full resource security checking is applied to the user. The user must reconnect to get the correct RESLEVEL processing.

z/OS User IDs for security checking on z/OS

IBM MQ initiates security checks based on user IDs associated with users, terminals, applications, and other resources. This collection of topics lists which user IDs are used for each type of security check.

z/OS *User IDs for connection security*

The user ID used for connection security depends on the type of connection.

Connection type	**User ID contents**
Batch connection	The user ID of the connecting task. For example: • The TSO user ID • The user ID assigned to a batch job by the USER JCL parameter • The user ID assigned to a started task by the STARTED class or the started procedures table
CICS connection	The CICS address space user ID.
IMS connection	The IMS region address space user ID.
Channel initiator connection	The channel initiator address space user ID.

z/OS *User IDs for command and command resource security*

The user ID used for command security or command resource security depends on where the command is issued from.

Issued from...	**User ID contents**
CSQINP1, CSQINP2, or CSQINPT	No check is made.
System command input queue	The user ID found in the *UserIdentifier* of the message descriptor of the message that contains the command. If the message does not contain a *UserIdentifier*, a user ID of blanks is passed to the security manager.
Console	The user ID signed onto the console. If the console is not signed on, the default user ID set by the CMDUSER system parameter in CSQ6SYSP. To issue commands from a console, the console must have the z/OS SYS AUTHORITY attribute.
SDSF/TSO console	TSO or job user ID.

Issued from...	**User ID contents**
Operations and control panels	TSO user ID. If you are going to use the operations and control panels, you must have the appropriate authority to issue the commands corresponding to the actions that you choose. In addition, you must have READ access to all the hlq.DISPLAY. *object* profiles in the MQCMDS class because the panels use the various DISPLAY commands to gather the information that they present.
MGCRE	If MGCRE is used with UTOKEN, the user ID in the UTOKEN. If MGCRE is issued without the UTOKEN, the TSO or job user ID is used.
CSQ0UTIL	Job user ID.
CSQUTIL	Job user ID.
CSQINPX	User ID of the channel initiator address space.

z/OS *User IDs for resource security (MQOPEN, MQSUB, and MQPUT1)*

This information shows the contents of the user IDs for normal and alternate user IDs for each type of connection. The number of checks is defined by the RESLEVEL profile. The user ID checked is that used for **MQOPEN**, **MQSUB**, or **MQPUT1** calls.

Note: All user ID fields are checked exactly as they are received. No conversions take place, and, for example, three user ID fields containing "Bob", "BOB", and "bob" are not equivalent.

z/OS *User IDs checked for batch connections*

The user ID checked for a batch connection depends on how the task is run and whether an alternate user ID has been specified.

Table 57. User ID checking against profile name for batch connections

Alternate user ID specified on open?	**hlq.ALTERNATE.USER.userid profile**	**hlq.CONTEXT.queuename profile**	**hlq.resourcename profile**
No	-	JOB	JOB
Yes	JOB	JOB	ALT

Key:

ALT
: Alternate user ID.

JOB

- The user ID of a TSO or z/OS UNIX System Services sign-on.
- The user ID assigned to a batch job.
- The user ID assigned to a started task by the STARTED class or the started procedures table.
- The user ID associated with the executing Db2 stored procedure

A Batch job is performing an MQPUT1 to a queue called Q1 with RESLEVEL set to READ and alternate user ID checking turned off.

Checks made at different RACF(r) access levels for batch connections and User ID checking against profile name for batch connections show that the job user ID is checked against profile hlq.Q1.

z/OS *User IDs checked for CICS connections*

The user IDs checked for CICS connections depend on whether one or two checks are to be carried out, and whether an alternate user ID is specified.

Table 58. User ID checking against profile name for CICS-type user IDs

Alternate user ID specified on open?	hlq.ALTERNATE.USER.userid profile	hlq.CONTEXT.queuename profile	hlq.resourcename profile
No, 1 check	-	ADS	ADS
No, 2 checks	-	ADS+TXN	ADS+TXN
Yes, 1 check	ADS	ADS	ADS
Yes, 2 checks	ADS+TXN	ADS+TXN	ADS+ALT

Key:

ALT
Alternate user ID

ADS
The user ID associated with the CICS batch job or, if CICS is running as a started task, through the STARTED class or the started procedures table.

TXN
The user ID associated with the CICS transaction. This is normally the user ID of the terminal user who started the transaction. It can be the CICS DFLTUSER, a PRESET security terminal, or a manually signed-on user.

Determine the user IDs checked for the following conditions:

- The RACF access level to the RESLEVEL profile, for a CICS address space user ID, is set to NONE.
- An `MQOPEN` call is made against a queue with MQOO_OUTPUT and MQOO_PASS_IDENTITY_CONTEXT.

First, see how many CICS user IDs are checked based on the CICS address space user ID access to the RESLEVEL profile. From Table 53 on page 231 in topic "RESLEVEL and CICS connections" on page 230, two user IDs are checked if the RESLEVEL profile is set to NONE. Then, from Table 58 on page 236 on, these checks are carried out:

- The hlq.ALTERNATE.USER.userid profile is not checked.
- The hlq.CONTEXT.queuename profile is checked with both the CICS address space user ID and the CICS transaction user ID.
- The hlq.resourcename profile is checked with both the CICS address space user ID and the CICS transaction user ID.

This means that four security checks are made for this `MQOPEN` call.

z/OS *User IDs checked for IMS connections*

The user IDs checked for IMS connections depend on whether one or two checks are to be performed, and whether an alternate user ID is specified. If a second user ID is checked, it depends on the type of dependent region and on which user IDs are available.

Table 59. User ID checking against profile name for IMS-type user IDs

Alternate user ID specified on open?	hlq.ALTERNATE.USER.userid profile	hlq.CONTEXT.queuename profile	hlq.resourcename profile
No, 1 check	-	REG	REG
No, 2 checks	-	REG+SEC	REG+SEC
Yes, 1 check	REG	REG	REG

Table 59. User ID checking against profile name for IMS-type user IDs (continued)

Alternate user ID specified on open?	hlq.ALTERNATE.USER.userid profile	hlq.CONTEXT.queuename profile	hlq.resourcename profile
Yes, 2 checks	REG+SEC	REG+SEC	REG+ALT

Key:

ALT
Alternate user ID.

REG
The user ID is normally set through the STARTED class or the started procedures table or, if IMS is running, from a submitted job, by the USER JCL parameter.

SEC
The second user ID is associated with the work being done in a dependent region. It is determined according to Table 60 on page 237.

Table 60. How the second user ID is determined for the IMS connection

Types of dependent region	Hierarchy for determining the second user ID
• BMP message driven and successful GET UNIQUE issued. • IFP and GET UNIQUE issued. • MPP.	User ID associated with the IMS transaction if the user is signed on. LTERM name if available. PSBNAME.
• BMP message driven and successful GET UNIQUE not issued. • BMP not message driven. • IFP and GET UNIQUE not issued.	User ID associated with the IMS dependent region address space if this is not all blanks or all zeros. PSBNAME.

z/OS *User IDs used by the channel initiator*
This collection of topics describes the user IDs used and checked for receiving channels and for client MQI requests issued over server-connection channels. Information is provided for TCP/IP and for LU6.2

You can use the PUTAUT parameter of the receiving channel definition to determine the type of security checking used. To get consistent security checking throughout your IBM MQ network, you can use the ONLYMCA and ALTMCA options.

You can use the DISPLAY CHSTATUS command to determine the user identifier used by the MCA.

z/OS *Receiving channels using TCP/IP*
The user IDs checked depend on the PUTAUT option of the channel and on whether one or two checks are to be performed.

Table 61. User IDs checked against profile name for TCP/IP channels

PUTAUT option specified on receiver or requester channel	hlq.ALTERNATE.USER.userid profile	hlq.CONTEXT.queuename profile	hlq.resourcename profile
DEF, 1 check	-	CHL	CHL
DEF, 2 checks	-	CHL + MCA	CHL + MCA
CTX, 1 check	CHL	CHL	CHL

Table 61. User IDs checked against profile name for TCP/IP channels (continued)

PUTAUT option specified on receiver or requester channel	**hlq.ALTERNATE.USER.userid profile**	**hlq.CONTEXT.queuename profile**	**hlq.resourcename profile**
CTX, 2 checks	CHL + MCA	CHL + MCA	CHL + ALT
ONLYMCA, 1 check	-	MCA	MCA
ONLYMCA, 2 checks	-	MCA	MCA
ALTMCA, 1 check	MCA	MCA	MCA
ALTMCA, 2 checks	MCA	MCA	MCA + ALT

Key:

MCA (MCA user ID)
The user ID specified for the MCAUSER channel attribute at the receiver; if blank, the channel initiator address space user ID of the receiver or requester side is used.

CHL (Channel user ID)
On TCP/IP, security is not supported by the communication system for the channel. If Transport Layer Security (TLS) is being used and a digital certificate has been flowed from the partner, the user ID associated with this certificate (if installed), or the user ID associated with a matching filter found by using RACF Certificate Name Filtering (CNF), is used. If no associated user ID is found, or if TLS is not being used, the user ID of the channel initiator address space of the receiver or requester end is used as the channel user ID on channels defined with the PUTAUT parameter set to DEF or CTX.

Note: The use of RACF Certificate Name Filtering (CNF) allows you to assign the same RACF user ID to multiple remote users, for example all the users in the same organization unit, who would naturally all have the same security authority. This means that the server does not have to have a copy of the certificate of every possible remote user across the world, and greatly simplifies certificate management and distribution.

If the PUTAUT parameter is set to ONLYMCA or ALTMCA for the channel, the channel user ID is ignored and the MCA user ID of the receiver or requester is used. This also applies to TCP/IP channels using TLS.

ALT (Alternate user ID)
The user ID from the context information (that is, the *UserIdentifier* field) within the message descriptor of the message. This user ID is moved into the *AlternateUserID* field in the object descriptor before an **MQOPEN** or **MQPUT1** call is issued for the target destination queue.

z/OS *Receiving channels using LU 6.2*
The user IDs checked depend on the PUTAUT option of the channel and on whether one or two checks are to be performed.

Table 62. User IDs checked against profile name for LU 6.2 channels

PUTAUT option specified on receiver or requester channel	**hlq.ALTERNATE.USER.userid profile**	**hlq.CONTEXT.queuename profile**	**hlq.resourcename profile**
DEF, 1 check	-	CHL	CHL
DEF, 2 checks	-	CHL + MCA	CHL + MCA
CTX, 1 check	CHL	CHL	CHL
CTX, 2 checks	CHL + MCA	CHL + MCA	CHL + ALT

Table 62. User IDs checked against profile name for LU 6.2 channels (continued)

PUTAUT option specified on receiver or requester channel	hlq.ALTERNATE.USER.userid profile	hlq.CONTEXT.queuename profile	hlq.resourcename profile
ONLYMCA, 1 check	-	MCA	MCA
ONLYMCA, 2 checks	-	MCA	MCA
ALTMCA, 1 check	MCA	MCA	MCA
ALTMCA, 2 checks	MCA	MCA	MCA + ALT

Key:

MCA (MCA user ID)
The user ID specified for the MCAUSER channel attribute at the receiver; if blank, the channel initiator address space user ID of the receiver or requester side is used.

CHL (Channel user ID)

Requester-server channels
If the channel is started from the requester, there is no opportunity to receive a network user ID (the channel user ID).

If the PUTAUT parameter is set to DEF or CTX on the requester channel, the channel user ID is that of the channel initiator address space of the requester because no user ID is received from the network.

If the PUTAUT parameter is set to ONLYMCA or ALTMCA, the channel user ID is ignored and the MCA user ID of the requester is used.

Other channel types
If the PUTAUT parameter is set to DEF or CTX on the receiver or requester channel, the channel user ID is the user ID received from the communications system when the channel is initiated.

- If the sending channel is on z/OS, the channel user ID received is the channel initiator address space user ID of the sender.
- If the sending channel is on a different platform (for example, AIX), the channel user ID received is typically provided by the USERID parameter of the channel definition.

If the user ID received is blank, or no user ID is received, a channel user ID of blanks is used.

ALT (Alternate user ID)
The user ID from the context information (that is, the *UserIdentifier* field) within the message descriptor of the message. This user ID is moved into the *AlternateUserID* field in the object descriptor before an MQOPEN or MQPUT1 call is issued for the target destination queue.

z/OS *Client MQI requests*

Various user IDs can be used, depending on which user IDs and environment variables have been set. These user IDs are checked against various profiles, depending on the PUTAUT option used and whether an alternate user ID is specified.

This section describes the user IDs checked for client MQI requests issued over server-connection channels for TCP/IP and LU 6.2. The MCA user ID and channel user ID are as for the TCP/IP and LU 6.2 channels described in the previous sections.

For server-connection channels, the user ID received from the client is used if the MCAUSER attribute is blank.

See "Access control for clients" on page 102 for more information.

For client **MQOPEN**, **MQSUB**, and **MQPUT1** requests, use the following rules to determine the profile that is checked:

- If the request specifies alternate-user authority, a check is made against the *hlq*.ALTERNATE.USER. *userid* profile.
- If the request specifies context authority, a check is made against the *hlq*.CONTEXT. *queuename* profile.
- For all **MQOPEN**, **MQSUB**, and **MQPUT1** requests, a check is made against the *hlq.resourcename* profile.

When you have determined which profiles are checked, use the following table to determine which user IDs are checked against these profiles.

Table 63. User IDs checked against profile name for LU 6.2 and TCP/IP server-connection channels

PUTAUT option specified on server-connection channel	Alternate user ID specified on open?	hlq.ALTERNATE.USER.userid profile	hlq.CONTEXT.queuename profile	hlq.resourcename profile
DEF, 1 check	No	-	CHL	CHL
DEF, 1 check	Yes	CHL	CHL	CHL
DEF, 2 checks	No	-	CHL + MCA	CHL + MCA
DEF, 2 checks	Yes	CHL + MCA	CHL + MCA	CHL + ALT
ONLYMCA, 1 check	No	-	MCA	MCA
ONLYMCA, 1 check	Yes	MCA	MCA	MCA
ONLYMCA, 2 checks	No	-	MCA	MCA
ONLYMCA, 2 checks	Yes	MCA	MCA	MCA + ALT

Key:

MCA (MCA user ID)
The user ID specified for the MCAUSER channel attribute at the server-connection; if blank, the channel initiator address space user ID is used.

CHL (Channel user ID)
On TCP/IP, security is not supported by the communication system for the channel. If Transport Layer Security (TLS) is being used and a digital certificate has been flowed from the partner, the user ID associated with this certificate (if installed), or the user ID associated with a matching filter found by using RACF Certificate Name Filtering (CNF), is used. If no associated user ID is found, or if TLS is not being used, the user ID of the channel initiator address space is used as the channel user ID on channels defined with the PUTAUT parameter set to DEF or CTX.

Note: The use of RACF Certificate Name Filtering (CNF) allows you to assign the same RACF user ID to multiple remote users, for example all the users in the same organization unit, who would naturally all have the same security authority. This means that the server does not have to have a copy of the certificate of every possible remote user across the world, and greatly simplifies certificate management and distribution.

If the PUTAUT parameter is set to ONLYMCA or ALTMCA for the channel, the channel user ID is ignored and the MCA user ID of the server-connection channel is used. This also applies to TCP/IP channels using TLS.

ALT (Alternate user ID)
The user ID from the context information (that is, the *UserIdentifier* field) within the message descriptor of the message. This user ID is moved into the *AlternateUserID* field in the object or subscription descriptor before an **MQOPEN**, **MQSUB** or **MQPUT1** call is issued on behalf of the client application.

z/OS *Channel initiator example*

An example of how user IDs are checked against RACF profiles.

A user performs an **MQPUT1** operation to a queue on queue manager QM01 that resolves to a queue called QB on queue manager QM02. The message is sent on a TCP/IP channel called QM01.TO.QM02. RESLEVEL is set to NONE, and the open is performed with alternate user ID and context checking. The receiver channel definition has PUTAUT(CTX) and the MCA user ID is set. Which user IDs are used on the receiving channel to put the message to queue QB?

Answer: Table 55 on page 232 shows that two user IDs are checked because RESLEVEL is set to NONE.

Table 61 on page 237 shows that, with PUTAUT set to CTX and 2 checks, the following user IDs are checked:

- The channel initiator user ID and the MCAUSER user ID are checked against the hlq.ALTERNATE.USER.userid profile.
- The channel initiator user ID and the MCAUSER user ID are checked against the hlq.CONTEXT.queuename profile.
- The channel initiator user ID and the alternate user ID specified in the message descriptor (MQMD) are checked against the hlq.Q2 profile.

z/OS *User IDs used by the intra-group queuing agent*

The user IDs that are checked when the intra-group queuing agent opens destination queues are determined by the values of the **IGQAUT** and **IGQUSER** queue manager attributes.

The possible user IDs are:

Intra-group queuing user ID (IGQ)
The user ID determined by the **IGQUSER** attribute of the receiving queue manager. If this is set to blanks, the user ID of the receiving queue manager is used. However, because the receiving queue manager has authority to access all queues defined to it, security checks are not performed for the receiving queue manager's user ID. In this case:

- If only one user ID is to be checked and the user ID is that of the receiving queue manager, no security checks take place. This can occur when **IGQAUT** is set to `ONLYIGQ` or `ALTIGQ`.
- If two user IDs are to be checked and one of the user IDs is that of the receiving queue manager, security checks take place for the other user ID only. This can occur when **IGQAUT** is set to `DEF`, `CTX`, or `ALTIGQ`.
- If two user IDs are to be checked and both user IDs are that of the receiving queue manager, no security checks take place. This can occur when **IGQAUT** is set to `ONLYIGQ`.

Sending queue manager user ID (SND)
The user ID of the queue manager within the queue sharing group that put the message on to the SYSTEM.QSG.TRANSMIT.QUEUE.

Alternate user ID (ALT)
The user ID specified in the *UserIdentifier* field in the message descriptor of the message.

Table 64. User IDs checked against profile name for intra-group queuing			
IGQAUT option specified on receiving queue manager	**hlq.ALTERNATE.USER.userid profile**	**hlq.CONTEXT.queuename profile**	**hlq.resourcename profile**
DEF, 1 check	-	SND	SND
DEF, 2 checks	-	SND +IGQ	SND +IGQ
CTX, 1 check	SND	SND	SND
CTX, 2 checks	SND + IGQ	SND +IGQ	SND + ALT
ONLYIGQ, 1 check	-	IGQ	IGQ
ONLYIGQ, 2 checks	-	IGQ	IGQ
ALTIGQ, 1 check	-	IGQ	IGQ
ALTIGQ, 2 checks	IGQ	IGQ	IGQ + ALT

Key:

ALT
Alternate user ID.

IGQ
IGQ user ID.

SND
Sending queue manager user ID.

z/OS *Blank user IDs and UACC levels*

If a blank user ID occurs, a RACF undefined user is signed on. Do not grant wide-ranging access to the undefined user.

Blank user IDs can exist when a user is manipulating messages using context or alternate-user security, or when IBM MQ is passed a blank user ID. For example, a blank user ID is used when a message is written to the system-command input queue without context.

Note: A user ID of " * " (that is, an asterisk character followed by seven spaces) is treated as an undefined user ID.

IBM MQ passes the blank user ID to RACF and a RACF undefined user is signed on. All security checks then use the universal access (UACC) for the relevant profile. Depending on how you have set your access levels, the UACC might give the undefined user a wide-ranging access.

For example, if you issue this RACF command from TSO:

```
RDEFINE MQQUEUE Q.AVAILABLE.TO.EVERYONE UACC(UPDATE)
```

you define a profile that enables both z/OS-defined user IDs (that have not been put in the access list) and the RACF undefined user ID to put messages on, and get messages from, that queue.

To protect against blank user IDs you must plan your access levels carefully, and limit the number of people who can use context and alternate-user security. You must prevent people using the RACF undefined user ID from getting access to resources that they must not access. However, at the same time, you must allow access to people with defined user IDs. To do this, you can specify a user ID of asterisk (*) in a RACF command PERMIT, giving access to resources for all defined user IDs. Therefore all undefined user IDs (such as " * ") are denied access. For example, these RACF commands prevent the RACF undefined user ID from gaining access to the queue to put or get messages:

```
RDEFINE MQQUEUE Q.AVAILABLE.TO.RACF.DEFINED.USERS.ONLY UACC(NONE)
PERMIT Q.AVAILABLE.TO.RACF.DEFINED.USERS.ONLY CLASS(MQQUEUE) ACCESS(UPDATE) ID(*)
```

z/OS z/OS user IDs and Multi-Factor Authentication (MFA)

IBM Multi-Factor Authentication for z/OS allows z/OS security administrators to enhance SAF authentication, by requiring identified users to use multiple authentication factors (for example, both a password and a cryptographic token) to sign on to a z/OS system. IBM MFA also provides support for time-based one time password generation technologies such as RSA SecureId.

For the most part, IBM MQ is unaware of how users have "logged on" to the CICS or batch systems that are driving IBM MQ work, the signed on user ID credential is associated with the z/OS task or address space and IBM MQ uses this for checking authorization to resources. User IDs enabled for MFA can be used for authorization to IBM MQ resources and authentication through pass tickets used with the CICS and IMS bridges.

Important: Special considerations apply however, when using applications, such as the IBM MQ Explorer, which pass a user ID and password credentials on an MQCONNX API call with the *MQCSP_AUTH_USER_ID_AND_PWD* option. IBM MQ has no facility to pass an additional credential on this API request.

Limitations and potential workarounds are described in the following text.

IBM MQ Explorer

The IBM MQ Explorer cannot be used to log on to a z/OS system with a userid for which MFA is enabled because there is no facility for passing a second authentication factor from the IBM MQ Explorer to z/OS.

Additionally, there are two different mechanisms used by the IBM MQ Explorer to re-use a user ID and password credential, that need special attention when one time use passwords are in effect:

1. IBM MQ Explorer has the capability to store passwords in an obfuscated format on the local machine for login at a later time. This capability must be disabled by having explorer prompt for a password each time a connection is made to the z/OS queue manager.

   To do this, use the following procedure:

   a. Select **Queue Managers**.
   b. From the list displayed, choose the queue manager you require and right click that queue manager.
   c. Select **Connection Details** from the menu list that appears.
   d. Select **Properties** from the next menu list and choose the **Userid** tab.

      Ensure that you select the **prompt for password** radio button.
2. Various operations in the IBM MQ Explorer, such as browsing messages on queues, testing subscriptions, and so on, start a new thread which authenticates to IBM MQ using the credential first used at logon. Since the password credential cannot be re-used, you cannot use these operations.

There are two possible workarounds at the MFA configuration level for these issues:

- Use the application ID exclusion of MFA to exclude the IBM MQ tasks from MFA processing altogether.

  To do this, issue the following commands:

  1.
     ```
     RDEFINE MFADEF MFABYPASS.USERID.chinuser
     ```

     where *chinuser* is the channel initiator address space level user Id (associated with the channel initiator through the STC class)

  2.
     ```
     PERMIT MFABYPASS.USERID.chinuser CLASS MFADEF ACCESS(READ) ID(explorer user)
     ```

For more information on this approach, see Bypassing IBM MFA for applications.

- Use Out-of-band support on MFA, which was introduced with IBM MFA 1.2. With this approach, you pre-authenticate to the IBM MFA web server, and in addition to your user ID and password, specify additional authentication as determined through the policy. IBM MFA server generates a cache token credential that you then specify on the IBM MQ Explorer authentication dialogue. The security administrator can allow this credential to be replayed for a reasonable period of time, so enabling normal IBM MQ Explorer use.

For more information on this approach see Introduction to IBM MFA.

z/OS IBM MQ for z/OS security management

IBM MQ uses an in-storage table to hold information relating to each user and the access requests made by each user. To manage this table efficiently and to reduce the number of requests made from IBM MQ to the external security manager (ESM), a number of controls are available.

These controls are available through both the operations and control panels and IBM MQ commands.

z/OS *User ID reverification*

If the RACF definition of a user who is using IBM MQ resources has been changed, for example by connecting the user to a new group, you can tell the queue manager to sign this user on again the next time it tries to access an IBM MQ resource. You can do this by using the IBM MQ command RVERIFY SECURITY.

- User HX0804 is getting and putting messages to the PAYROLL queues on queue manager PRD1. However HX0804 now requires access to some of the PENSION queues on the same queue manager (PRD1).
- The data security administrator connects user HX0804 to the RACF group that allows access to the PENSION queues.
- So that HX0804 can access the PENSION queues immediately (that is, without shutting down queue manager PRD1 or waiting for HX0804 to time out) you must use the IBM MQ command:

```
RVERIFY SECURITY(HX0804)
```

Note: If you turn off user ID timeout for long periods of time (days or even weeks) while the queue manager is running, you must remember to run the RVERIFY SECURITY command for any users that have been revoked or deleted in that time.

z/OS *User ID timeouts*

You can make IBM MQ sign a user off a queue manager after a period of inactivity.

When a user accesses an IBM MQ resource, the queue manager tries to sign this user on to the queue manager (if subsystem security is active). This means that the user is authenticated to the ESM. This user remains signed on to IBM MQ until either the queue manager is shut down, or until the user ID is *timed out* (the authentication lapses) or reverified (reauthenticated).

When a user is timed out, the user ID is *signed off* within the queue manager and any security-related information retained for this user is discarded. The signing on and off of the user within the queue manager is not apparent to the application program or to the user.

Users are eligible for timeout when they have not used any IBM MQ resources for a predetermined amount of time. This time period is set by the MQSC ALTER SECURITY command.

Two values can be specified in the ALTER SECURITY command:

TIMEOUT
: The time period in minutes that an unused user ID and its associated resources can remain within the IBM MQ queue manager.

INTERVAL
The time period in minutes between checks for user IDs and their associated resources, to determine whether the *TIMEOUT* has expired.

For example, if the *TIMEOUT* value is 30 and the *INTERVAL* value is 10, every 10 minutes IBM MQ checks user IDs and their associated resources to determine whether any have not been used for 30 minutes. If a timed-out user ID is found, that user ID is signed off within the queue manager. If any timed-out resource information associated with non-timed-out user IDs is found, that resource information is discarded. If you do not want to time out user IDs, set the *INTERVAL* value to zero. However, if the *INTERVAL* value is zero, storage occupied by user IDs and their associated resources is not freed until you issue a **REFRESH SECURITY** or **RVERIFY SECURITY** command.

Tuning this value can be important if you have many one-off users. If you set small interval and timeout values, resources that are no longer required are freed.

Note: If you use values for *INTERVAL* or *TIMEOUT* other than the defaults, you must reenter the command at every queue manager startup. You can do this automatically by putting the **ALTER SECURITY** command in the CSQINP1 data set for that queue manager.

z/OS *Refreshing queue manager security on z/OS*

IBM MQ for z/OS caches RACF data to improve performance. When you change certain security classes, you must refresh this cached information. Refresh security infrequently, for performance reasons. You can also choose to refresh only TLS security information.

When a queue is opened for the first time (or for the first time since a security refresh) IBM MQ performs a RACF check to obtain the user's access rights and places this information in the cache. The cached data includes user IDs and resources on which security checking has been performed. If the queue is opened again by the same user, the presence of the cached data means that IBM MQ does not have to issue RACF checks, which improves performance. The action of a security refresh is to discard any cached security information and so force IBM MQ to make a new check against RACF. Whenever you add, change or delete a RACF resource profile that is held in the MQADMIN, MXADMIN, MQPROC, MXPROC, MQQUEUE, MXQUEUE, MQNLIST, MXNLIST, or MXTOPIC class, you must tell the queue managers that use this class to refresh the security information that they hold. To do this, issue the following commands:

- The RACF SETROPTS RACLIST(classname) REFRESH command to refresh at the RACF level.
- The IBM MQ REFRESH SECURITY command to refresh the security information held by the queue manager. This command needs to be issued by each queue manager that accesses the profiles that have changed. If you have a queue sharing group, you can use the command scope attribute to direct the command to all the queue managers in the group.

  Note: If you have connected a new user to an existing group, you need to run the IBM MQ RVERIFY SECURITY(userid) command. The REFRESH SECURITY(*) command does not let the queue manager sign this user on again, the next time it tries to access an IBM MQ resource.

If you are using generic profiles in any of the IBM MQ classes, you must also issue normal RACF refresh commands if you change, add, or delete any generic profiles. For example, SETROPTS GENERIC(classname) REFRESH.

However, if a RACF resource profile is added, changed or deleted, and the resource to which it applies has not yet been accessed (so no information is cached), IBM MQ uses the new RACF information without a REFRESH SECURITY command being issued.

If RACF auditing is turned on, (for example, by using the RACF RALTER AUDIT(access-attempt (audit_access_level)) command), no caching takes place, and therefore IBM MQ refers directly to the RACF dataspace for every check. Changes are therefore picked up immediately and REFRESH SECURITY is not necessary to access the changes. You can confirm whether RACF auditing is on by using the RACF RLIST command. For example, you could issue the command

```
RLIST MQQUEUE (qmgr.SYSTEM.COMMAND.INPUT) GEN
```

and receive the results

```
CLASS      NAME
-----      ----
MQQUEUE    QP*.SYSTEM.COMMAND.*.** (G)
    AUDITING
    --------
    FAILURES(READ)
```

This indicates that auditing is set on. For more information, see the *z/OS Security Server RACF Auditor's Guide* and the *z/OS Security Server RACF Command Language Reference*.

Figure 17 on page 246 summarizes the situations in which security information is cached and in which cached information is used.

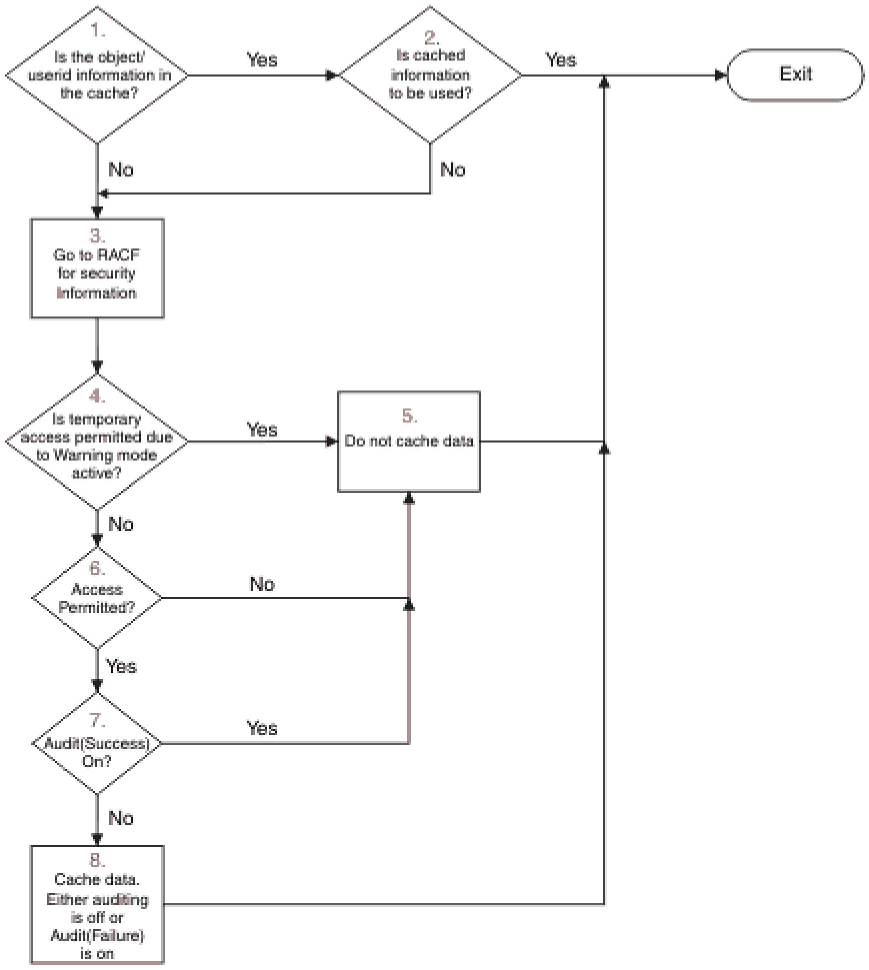

1. Is the object/ userid information in the cache?
Yes
2. Is cached information to be used?
Yes
Exit
No
No
3. Go to RACF for security Information
4. Is temporary access permitted due to Warning mode active?
Yes
5. Do not cache data
No
6. Access Permitted?
No
Yes
7. Audit(Success) On?
Yes
No
8. Cache data. Either auditing is off or Audit(Failure) is on

Figure 17. Logic flow for IBM MQ security caching

If you change your security settings by adding or deleting switch profiles in the MQADMIN or MXADMIN classes, use one of these commands to pick up these changes dynamically:

REFRESH SECURITY(*)
REFRESH SECURITY(MQADMIN)
REFRESH SECURITY(MXADMIN)

This means you can activate new security types, or deactivate them without having to restart the queue manager.

For performance reasons, these are the only classes affected by the REFRESH SECURITY command. You do not need to use REFRESH SECURITY if you change a profile in either the MQCONN or MQCMDS classes.

Note: A refresh of the MQADMIN or MXADMIN class is not required if you change a RESLEVEL security profile.

For performance reasons, use REFRESH SECURITY as infrequently as possible, ideally at off-peak times. You can minimize the number of security refreshes by connecting users to RACF groups that are already in the access list for IBM MQ profiles, rather than putting individual users in the access lists. In this way, you change the user rather than the resource profile. You can also RVERIFY SECURITY the appropriate user instead of refreshing security.

As an example of REFRESH SECURITY, suppose you define the new profiles to protect access to queues starting with INSURANCE.LIFE on queue manager PRMQ. You use these RACF commands:

```
RDEFINE MQQUEUE PRMQ.INSURANCE.LIFE.** UACC(NONE)
PERMIT PRMQ.INSURANCE.LIFE.** ID(LIFEGRP) ACCESS(UPDATE)
```

You must issue the following command to tell RACF to refresh the security information that it holds, for example:

```
SETROPTS RACLIST(MQQUEUE) REFRESH
```

Because these profiles are generic, you must tell RACF to refresh the generic profiles for MQQUEUE. For example:

```
SETROPTS GENERIC(MQQUEUE) REFRESH
```

Then you must use this command to tell queue manager PRMQ that the queue profiles have changed:

```
REFRESH SECURITY(MQQUEUE)
```

Refreshing SSL/TLS security

To refresh the cached view of the TLS Key Repository, issue the REFRESH SECURITY command with the option TYPE(SSL). This enables you to update some of your TLS settings without having to restart your channel initiator.

z/OS *Displaying security status*

To display the status of the security switches, and other security controls, issue the MQSC DISPLAY SECURITY command.

The following figure shows typical output of the DISPLAY SECURITY ALL command.

```
CSQH015I +CSQ1 Security timeout = 54 MINUTES
CSQH016I +CSQ1 Security interval = 12 MINUTES
CSQH030I +CSQ1 Security switches ...
CSQH034I +CSQ1 SUBSYSTEM: ON, 'SQ05.NO.SUBSYS.SECURITY' not found
CSQH032I +CSQ1 QMGR: ON, 'CSQ1.YES.QMGR.CHECKS' found
CSQH031I +CSQ1 QSG: OFF, 'SQ05.NO.QSG.CHECKS' found
CSQH031I +CSQ1 CONNECTION: OFF, 'CSQ1.NO.CONNECT.CHECKS' found
CSQH034I +CSQ1 COMMAND: ON, 'CSQ1.NO.COMMAND.CHECKS' not found
CSQH031I +CSQ1 CONTEXT: OFF, 'CSQ1.NO.CONTEXT.CHECKS' found
CSQH034I +CSQ1 ALTERNATE USER: ON, 'CSQ1.NO.ALTERNATE.USER.CHECKS' not found
CSQH034I +CSQ1 PROCESS: ON, 'CSQ1.NO.PROCESS.CHECKS' not found
CSQH034I +CSQ1 NAMELIST: ON, 'CSQ1.NO.NLIST.CHECKS' not found
CSQH034I +CSQ1 QUEUE: ON, 'CSQ1.NO.QUEUE.CHECKS' not found
CSQH034I +CSQ1 TOPIC: ON, 'CSQ1.NO.TOPIC.CHECKS' not found
CSQH031I +CSQ1 COMMAND RESOURCES: OFF, 'CSQ1.NO.CMD.RESC.CHECKS' found
CSQ9022I +CSQ1 CSQHPDTC ' DISPLAY SECURITY' NORMAL COMPLETION
```

Figure 18. Typical output from the DISPLAY SECURITY command

The example shows that the queue manager that replied to the command has subsystem, command, alternate user, process, namelist, and queue security active at queue manager level but not at queue sharing group level. Connection, command resource, and context security are not active. It also shows that user ID timeouts are active, and that every 12 minutes the queue manager checks for user IDs that have not been used in this queue manager for 54 minutes and removes them.

Note: This command shows the current security status. It does not necessarily reflect the current status of the switch profiles defined to RACF, or the status of the RACF classes. For example, the switch profiles might have been changed since the last restart of this queue manager or REFRESH SECURITY command.

z/OS Security installation tasks for z/OS

After installing and customizing IBM MQ, authorize started task procedures to RACF, authorize access to various resources, and set up RACF definitions. Optionally, configure your system for TLS.

When IBM MQ is first installed and customized, you must perform these security-related tasks:

1. Set up IBM MQ data set and system security by:
    - Authorizing the queue manager started-task procedure xxxxMSTR and the distributed queuing started-task procedure xxxxCHIN to run under RACF.
    - Authorizing access to queue manager data sets.
    - Authorizing access to resources for those user IDs that will use the queue manager and utility programs.
    - Authorizing access for those queue managers that will use the coupling facility list structures.
    - Authorizing access for those queue managers that will use Db2.
2. Set up RACF definitions for IBM MQ security.
3. If you want to use Transport Layer Security (TLS), prepare your system to use certificates and keys.

z/OS *Setting up IBM MQ for z/OS data set security*

There are many types of IBM MQ user. Use RACF to control their access to system data sets.

The possible users of IBM MQ data sets include the following entities:

- The queue manager itself.
- The channel initiator
- IBM MQ administrators, who need to create IBM MQ data sets, run utility programs, and similar tasks.
- Application programmers who need to use the IBM MQ-supplied copybooks, include data sets, macros, and similar resources.
- Applications involving one or more of:
    - Batch jobs
    - TSO users
    - CICS regions
    - IMS regions
- Data sets CSQOUTX and CSQSNAP
- Dynamic queues SYSTEM.CSQXCMD.*

For all these potential users, protect the IBM MQ data sets with RACF.

You must also control access to all your 'CSQINP' data sets.

z/OS *RACF authorization of started-task procedures*

Some IBM MQ data sets are for the exclusive use of the queue manager. If you protect your IBM MQ data sets using RACF, you must also authorize the queue manager started-task procedure xxxxMSTR, and the distributed queuing started-task procedure xxxxCHIN, using RACF. To do this, use the STARTED class. Alternatively, you can use the started procedures table (ICHRIN03), but then you must perform an IPL of your z/OS system before the changes take effect.

For more information, see the *z/OS Security Server RACF System Programmer's Guide*.

The RACF user ID identified must have the required access to the data sets in the started-task procedure. For example, if you associate a queue manager started task procedure called CSQ1MSTR with the RACF user ID QMGRCSQ1, the user ID QMGRCSQ1 must have access to the z/OS resources accessed by the CSQ1 queue manager.

Also, the content of the GROUP field in the user ID of the queue manager must be the same as the content of the GROUP field in the STARTED profile for that queue manager. If the content in each GROUP field does not match then the appropriate user ID is prevented from entering the system. This situation causes IBM MQ to run with an undefined user ID and consequently close due to a security violation.

The RACF user IDs associated with the queue manager and channel initiator started task procedures must not have the TRUSTED attribute set.

z/OS *Authorizing access to data sets*
The IBM MQ data sets should be protected so that no unauthorized user can run a queue manager instance, or gain access to any queue manager data. To do this, use normal z/OS RACF data set protection.

Table 65 on page 249 summarizes the RACF access that the queue manager started task procedure must have to the different data sets.

Table 65. RACF access to data sets associated with a queue manager

RACF access	Data sets
READ	• `thlqual.SCSQAUTH` and `thlqual.SCSQANLx` (where x is the language letter for your national language). • The data sets referred to by CSQINP1, CSQINP2 and CSQXLIB in the queue manager's started task procedure. • SMDS data sets owned by other queue managers in the group. • Log, BSDS and archive log data sets for other queue managers in the group.
UPDATE	• All page sets and log and BSDS data sets. • SMDS data sets owned by a queue manager • SMDS data sets owned by other queue managers in the group, for the structures that the queue manager performs the RECOVER CFSTRUCT command.
ALTER	• All archive log data sets.

Table 66 on page 249 summarizes the RACF access that the started task procedure for distributed queuing must have to the different data sets.

Table 66. RACF access to data sets associated with distributed queuing

RACF access	Data sets
READ	• thlqual.SCSQAUTH, thlqual.SCSQANLx (where x is the language letter for your national language), and thlqual.SCSQMVR1. • LE library data sets. • The data sets referred to by CSQXLIB and CSQINPX in the channel initiator started task procedure.
UPDATE	• Data sets CSQOUTX and CSQSNAP

For more information, see the *z/OS Security Server RACF Security Administrator's Guide*.

z/OS *Encrypting data sets*

The IBM MQ data sets can be encrypted with z/OS data set encryption, so that the data is protected, or for regulatory reasons.

You can protect all page sets, active log, archive log, and bootstrap (BSDS) data sets with z/OS data set encryption.

Attention: You cannot protect shared message data sets (SMDS) with z/OS data set encryption by IBM MQ for z/OS 9.1.4 or earlier.

See the section, confidentiality for data at rest on IBM MQ for z/OS with data set encryption. for more information.

z/OS *Setting up IBM MQ for z/OS resource security*

There are many types of IBM MQ user. Use RACF to control their access to IBM MQ resources.

The possible users of IBM MQ resources, such as queues and channels include the following entities:

- The queue manager itself.
- The channel initiator
- IBM MQ administrators, who need to create IBM MQ data sets, run utility programs, and similar tasks
- Application programmers who need to use the IBM MQ-supplied copybooks, include data sets, macros, and similar resources.
- Applications involving one or more of:
  - Batch jobs
  - TSO users
  - CICS regions
  - IMS regions
- Data sets CSQOUTX and CSQSNAP
- Dynamic queues SYSTEM.CSQXCMD.*

For all these potential users, protect the IBM MQ resources with RACF. In particular, note that the channel initiator needs access to various resources, as described in "Security considerations for the channel initiator on z/OS" on page 256, and so the user ID under which it runs must be authorized to access these resources.

If you are using a queue sharing group, the queue manager might issue various commands internally, so the user ID it uses must be authorized to issue such commands. The commands are:

- DEFINE, ALTER, and DELETE for every object that has QSGDISP(GROUP)
- START and STOP CHANNEL for every channel used with CHLDISP(SHARED)

z/OS *Configuring your z/OS system to use TLS*

Use this topic as example of how to configure IBM MQ for z/OS with Transport Layer Security (TLS) using RACF commands.

If you want to use TLS for channel security, there are a number of tasks you need to perform on your system. (For details on using RACF commands for certificates and key repositories (key rings), see Working with TLS on z/OS .)

1. Create a key ring in RACF to hold all the keys and certificates for your system, using the RACF RACDCERT command. For example:

```
RACDCERT ID(CHINUSER) ADDRING(QM1RING)
```

The ID must be either the channel initiator address space user ID or the user ID you want to own the key ring if it is to be a shared key ring.

2. Create a digital certificate for each queue manager, using the RACF RACDCERT command.

   The label of the certificate must be either the value of the IBM MQ **CERTLABL** attribute, if it is set, or the default `ibmWebSphereMQ` with the name of the queue manager or queue sharing group appended. See Digital certificate labels for details. In this example it is `ibmWebSphereMQQM1`.

   For example:

```
RACDCERT ID(USERID) GENCERT
SUBJECTSDN(CN('username') O('IBM') OU('departmentname') C('England'))
WITHLABEL('ibmWebSphereMQQM1')
```

3. Connect the certificate in RACF to the key ring, using the RACF RACDCERT command. For example:

```
RACDCERT CONNECT(ID(USERID) LABEL('ibmWebSphereMQQM1') RING(QM1RING))
CONNECT ID(CHINUSER)
```

   You also need to connect any relevant signer certificates (from a certificate authority) to the key ring. That is, all certificate authorities for the TLS certificate of this queue manager and all certificate authorities for all TLS certificates that this queue manager communicates with. For example:

```
RACDCERT ID(CHINUSER)
CONNECT(CERTAUTH LABEL('My CA') RING(QM1RING) USAGE(CERTAUTH))
```

4. On each of your queue managers, use the IBM MQ ALTER QMGR command to specify the key repository that the queue manager needs to point to. For example, if the key ring is owned by the channel initiator address space:

```
ALTER QMGR SSLKEYR(QM1RING)
```

   or if you are using a shared key ring:

```
ALTER QMGR SSLKEYR(userid/QM1RING)
```

   where *userid* is the user ID that owns the shared key ring.

5. Certificate Revocation Lists (CRLs) allow the certificate authorities to revoke certificates that can no longer be trusted. CRLs are stored in LDAP servers. To access this list on the LDAP server, you first need to create an AUTHINFO object of AUTHTYPE CRLLDAP, using the IBM MQ DEFINE AUTHINFO command. For example:

```
DEFINE AUTHINFO(LDAP1)
AUTHTYPE(CRLLDAP)
CONNAME(ldap.server(389))
LDAPUSER('')
LDAPPWD('')
```

In this example, the certificate revocation list is stored in a public area of the LDAP server, so the LDAPUSER and LDAPPWD fields are not necessary.

Next, put your AUTHINFO object into a namelist, using the IBM MQ DEFINE NAMELIST command. For example:

```
DEFINE NAMELIST(LDAPNL) NAMES(LDAP1)
```

Finally, associate the namelist with each queue manager, using the IBM MQ ALTER QMGR command. For example:

```
ALTER QMGR SSLCRLNL(LDAPNL)
```

6. Set up your queue manager to run TLS calls, using the IBM MQ ALTER QMGR command. This defines server subtasks that handle SSL calls only, which leaves the normal dispatchers to continue processing as normal without being affected by any SSL calls. You must have at least two of these subtasks. For example:

   ```
   ALTER QMGR SSLTASKS(8)
   ```

   This change only takes effect when the channel initiator is restarted.
7. Specify the cipher specification to be used for each channel, using the IBM MQ DEFINE CHANNEL or ALTER CHANNEL command. For example:

   ```
   ALTER CHANNEL(LDAPCHL)
   CHLTYPE(SDR)
   SSLCIPH(TLS_RSA_WITH_AES_128_CBC_SHA256)
   ```

   Both ends of the channel must specify the same cipher specification.

z/OS Managing channel authentication records in a QSG

Channel authentication records apply to the queue manager that they are created on, they are not shared throughout the queue sharing group (QSG). Therefore if all the queue managers in the queue sharing group are required to have the same rules, some management needs to be carried out to keep all the rules the consistent.

1. Always add the `CMDSCOPE(*)` option to all `SET CHLAUTH` commands. This will send the command to all running queue managers in the queue sharing group
2. Use the `DISPLAY CHLAUTH` command with the `CMDSCOPE(*)` option and then analyze the responses to see if the records are the same from all queue managers. When an inconsistency is found a SET CHLAUTH command can be issued containing the same rule with `CMDSCOPE(*)` or `CMDSCOPE(`*`qmgr-name`*`)`.

3. Add a member to the queue manager's CSQINP2 concatenation (see Initialization commands for details) that has the full set of rules. These will be read as part of the queue manager's initialization process. If the `SET CHLAUTH` command uses `ACTION(ADD)` the rule will only be added if it didn't exist. Using `ACTION(REPLACE)` will replace an existing rule if it already exists or add it if it does not. The same member could then be placed in the CSQINP2 concatenation of all queue managers in the queue sharing group.
4. Use the CSQUTIL utility (see Issuing commands to IBM MQ (COMMAND) for details) to extract the rules from one queue manager using either the `MAKEDEF` or `MAKEREP` option. Then replay the output using CSQUTIL into the target queue manager.

Related concepts

Channel authentication records
To exercise more precise control over the access granted to connecting systems at a channel level, you can use channel authentication records.

z/OS Auditing considerations on z/OS

The normal RACF auditing controls are available for conducting a security audit of a queue manager. IBM MQ does not gather any security statistics of its own. The only statistics are those that can be created by auditing.

RACF auditing can be based upon:

- User IDs
- Resource classes
- Profiles

For more details, see the *z/OS Security Server RACF Auditor's Guide*.

Note: Auditing degrades performance; the more auditing you implement, the more performance is degraded. This is also a consideration for the use of the RACF WARNING option.

z/OS *Auditing RESLEVEL*

Use the RESAUDIT system parameter to control the production of RESLEVEL audit records. RACF GENERAL audit records are produced.

Produce RESLEVEL audit records by setting the RESAUDIT system parameter to YES. If the RESAUDIT parameter is set to NO, audit records are not produced. For more details about setting this parameter, see Using CSQ6SYSP.

If RESAUDIT is set to YES, no normal RACF audit records are taken when the RESLEVEL check is made to see what access an address space user ID has to the hlq.RESLEVEL profile. Instead, IBM MQ requests that RACF create a GENERAL audit record (event number 27). These checks are only carried out at connect time, so the performance cost is minimal.

⚠ **Attention:** RACFRW is no longer the suggested utility for processing RACF audit records. You should use the RACF SMF data unload utility as this is the preferred reporting method.

You can report the IBM MQ general audit records using the RACF report writer (RACFRW). You could use the following RACFRW commands to report the RESLEVEL access:

```
RACFRW
SELECT PROCESS
EVENT GENERAL
LIST
END
```

A sample report from RACFRW, excluding the *Date*, *Time*, and *SYSID* fields, is shown in Figure 19 on page 254.

```
       RACF REPORT - LISTING OF PROCESS RECORDS                                     PAGE   4
                                E
                                V  Q
                                E  U
*JOB/USER *STEP/  --TERMINAL--  N  A
   NAME    GROUP     ID    LVL  T  L

 WS21B    MQMGRP IGJZM000   0   27 0  JOBID=(WS21B 05.111 09:44:57),USERDATA=()
    TRUSTED USER                      AUTH=(NONE),REASON=(NONE)
                                 SESSION=TSOLOGON,TERMINAL=IGJZM000,
                                 LOGSTR='CSQH RESLEVEL CHECK PERFORMED AGAINST
PROFILE(QM66.RESLEVEL),
                                 CLASS(MQADMIN), ACCESS EQUATES TO
(CONTROL)',RESULT=SUCCESS,MQADMIN
```

Figure 19. Sample output from RACFRW showing RESLEVEL general audit records

From checking the LOGSTR data in this sample output, you can see that TSO user WS21B has CONTROL access to QM66.RESLEVEL. This means that all resource security checks are bypassed when user WS21B access QM66 resources.

For more information about using RACFRW, see The RACF report writer in the *z/OS Security Server RACF Auditor's Guide*.

z/OS Customizing security

If you want to change the way IBM MQ security operates, you must do this through the SAF exit (ICHRFR00), or exits in your external security manager.

To find out more about RACF exits, see the *z/OS Security Server RACROUTE Macro Reference* documentation.

Note: Because IBM MQ optimizes calls to the ESM, RACROUTE requests might not be made on, for example, every open for a particular queue by a particular user.

z/OS Security violation messages on z/OS

A security violation is indicated by the return code MQRC_NOT_AUTHORIZED in an application program or by a message in the job log.

A return code of MQRC_NOT_AUTHORIZED can be returned to an application program for the following reasons:

- A user is not allowed to connect to the queue manager. In this case, you get an ICH408I message in the Batch/TSO, CICS, or IMS job log.
- A user sign-on to the queue manager has failed because, for example, the job user ID is not valid or appropriate, or the task user ID or alternate user ID is not valid. One or more of these user IDs might not be valid because they have been revoked or deleted. In this case, you get an ICHxxxx message and possibly an IRRxxxx message in the queue manager job log giving the reason for the sign-on failure. For example:

```
ICH408I USER(NOTDFND ) GROUP(        ) NAME(???                 )
  LOGON/JOB INITIATION - USER AT TERMINAL          NOT RACF-DEFINED
IRR012I  VERIFICATION FAILED. USER PROFILE NOT FOUND
```

- An alternate user has been requested, but the job or task user ID does not have access to the alternate user ID. For this failure, you get a violation message in the job log of the relevant queue manager.
- A context option has been used or is implied by opening a transmission queue for output, but the job user ID or, where applicable, the task or alternate user ID does not have access to the context option. In this case, a violation message is put in the job log of the relevant queue manager.

- An unauthorized user has attempted to access a secured queue manager object, for example, a queue. In this case, an ICH408I message for the violation is put in the job log of the relevant queue manager. This violation might be due to the job or, when applicable, the task or alternate user ID.

Violation messages for command security and command resource security can also be found in the job log of the queue manager.

If the ICH408I violation message shows the queue manager jobname rather than a user ID, this is normally the result of a blank alternate user ID being specified. For example:

```
ICH408I JOB(MQS1MSTR) STEP(MQS1MSTR)
          MQS1.PAYROLL.REQUEST CL(MQQUEUE)
          INSUFFICIENT ACCESS AUTHORITY
          ACCESS INTENT(UPDATE )  ACCESS ALLOWED(NONE   )
```

You can find out who is allowed to use blank alternate user IDs by checking the access list of the MQADMIN profile hlq.ALTERNATE.USER.-BLANK-.

An ICH408I violation message can also be generated by:

- A command being sent to the system-command input queue without context. User-written programs that write to the system-command input queue should always use a context option. For more information, see "Profiles for context security" on page 214.
- When the job accessing the IBM MQ resource does not have a user ID associated with it, or when an IBM MQ adapter cannot extract the user ID from the adapter environment.

Violation messages might also be issued if you are using both queue sharing group and queue manager level security. You might get messages indicating that no profile has been found at queue manager level, but still be granted access because of a queue sharing group level profile.

```
ICH408I JOB(MQS1MSTR) STEP(MQS1MSTR)
          MQS1.PAYROLL.REQUEST CL(MQQUEUE)
          PROFILE NOT FOUND - REQUIRED FOR AUTHORITY CHECKING
          ACCESS INTENT(UPDATE )  ACCESS ALLOWED(NONE   )
```

See the z/OS for Security Server RACF Messages and Codes documentation for more information on ICH408I messages.

z/OS What to do if access is allowed or disallowed incorrectly

In addition to the information detailed in the z/OS documentation, use this checklist if access to a resource appears to be incorrectly controlled.

See the *z/OS Security Server RACF Security Administrator's Guide* for the detailed steps if access is allowed or disallowed.

- Are the switch profiles correctly set?
  - Is RACF active?
  - Are the IBM MQ RACF classes installed and active?

    Use the RACF command, SETROPTS LIST, to check this.
  - Use the IBM MQ DISPLAY SECURITY command to display the current switch status from the queue manager.
  - Check the switch profiles in the MQADMIN class.

    Use the RACF commands, SEARCH and RLIST, for this.
  - Recheck the RACF switch profiles by issuing the IBM MQ REFRESH SECURITY(MQADMIN) command.

- Has the RACF resource profile changed? For example, has universal access on the profile changed or has the access list of the profile changed?
  - Is the profile generic?

    If it is, issue the RACF command, SETROPTS GENERIC(classname) REFRESH.
  - Have you refreshed the security on this queue manager?

    If required, issue the RACF command SETROPTS RACLIST(classname) REFRESH.

    If required, issue the IBM MQ REFRESH SECURITY(*) command.
- Has the RACF definition of the user changed? For example, has the user been connected to a new group or has the user access authority been revoked?
  - Have you reverified the user by issuing the IBM MQ RVERIFY SECURITY(userid) command?
- Are security checks being bypassed due to RESLEVEL?
  - Check the connecting user ID's access to the RESLEVEL profile. Use the RACF audit records to determine what the RESLEVEL is set to.
  - For channels, remember that the access level that the channel initiator's userid has to RESLEVEL is inherited by all channels, so an access level, such as ALTER, that causes all checks to be bypassed causes security checks to be bypassed for all channels.
  - If you are running from CICS, check the transaction's RESSEC setting.
  - If RESLEVEL has been changed while a user is connected, they must disconnect and reconnect before the new RESLEVEL setting takes effect.
- Are you using queue sharing groups?
  - If you are using both queue sharing group and queue manager level security, check that you have defined all the correct profiles. If queue manager profile is not defined, a message is sent to the log stating that the profile was not found.
  - Have you used a combination of switch settings that is not valid so that full security checking has been set on?
  - Do you need to define security switches to override some of the queue sharing group settings for your queue manager?
  - Is a queue manager level profile taking precedence over a queue sharing group level profile?

z/OS Security considerations for the channel initiator on z/OS

If you are using resource security in a distributed queuing environment, the Channel initiator address space needs appropriate access to various IBM MQ resources. You can use the Integrated Cryptographic Support Facility (ICSF) to seed the password protection algorithm.

See the z/OS Cryptographic Services documentation for more information on ICSF.

Using resource security

If you are using resource security, consider the following points if you are using distributed queuing:

System queues
: The channel initiator address space needs RACF UPDATE access to the system queues listed at "System queue security" on page 204, and to all the user destination queues and the dead-letter queue (but see "Dead-letter queue security" on page 203).

Transmission queues
: The channel initiator address space needs ALTER access to all the user transmission queues.

Context security
: The channel user ID (and the MCA user ID if one has been specified) need RACF CONTROL access to the hlq.CONTEXT.queuename profiles in the MQADMIN class. Depending on the RESLEVEL profile, the channel user ID might also need CONTROL access to these profiles.

All channels need CONTROL access to the MQADMIN hlq.CONTEXT. dead-letter-queue profile. All channels (whether initiating or responding) can generate reports, and consequently they need CONTROL access to the hlq.CONTEXT.reply-q profile.

SENDER, CLUSSDR, and SERVER channels need CONTROL access to the hlq.CONTEXT.xmit-queue-name profiles since messages can be put onto the transmission queue to wake up the channel to end gracefully.

Note: If the channel user ID, or a RACF group to which the channel user ID is connected, has CONTROL or ALTER access to the hlq.RESLEVEL, then there are no resource checks for the channel initiator or any of its channels.

See "Profiles for context security" on page 214 "RESLEVEL and the channel initiator connection" on page 232 and "User IDs for security checking on z/OS" on page 234 for more information.

CSQINPX
If you are using the CSQINPX input data set, the channel initiator also needs READ access to CSQINPX, and UPDATE access to data set CSQOUTX and dynamic queues SYSTEM.CSQXCMD.*.

Connection security
The channel initiator address space connection requests use a connection type of CHIN, for which appropriate access security must be set, see "Connection security profiles for the channel initiator" on page 198.

Data sets
The channel initiator address space needs appropriate access to queue manager data sets, see "Authorizing access to data sets" on page 249.

Commands
The distributed queuing commands (for example, DEFINE CHANNEL, START CHINIT, START LISTENER, and other channel commands) must have appropriate command security set, see Table 49 on page 217.

If you are using a queue sharing group, the channel initiator might issue various commands internally, so the user ID it uses must be authorized to issue such commands. These commands are START and STOP CHANNEL for every channel used with CHLDISP(SHARED).

If the PSMODE of the queue manager is not DISABLED, the channel initiator must have READ access to the DISPLAY PUBSUB command.

Channel security
Channels, particularly receivers and server-connections, need appropriate security to be set up; see "User IDs for security checking on z/OS" on page 234 for more information.

You can also use the Transport Layer Security (TLS) protocol to provide security on channels. See "TLS security protocols in IBM MQ " on page 24 for more information about using TLS with IBM MQ.

See also "Access control for clients" on page 102 for information about server-connection security.

User IDs
The user IDs described in "User IDs used by the channel initiator" on page 237 and "User IDs used by the intra-group queuing agent" on page 241 need the following access:

- RACF UPDATE access to the appropriate destination queues and the dead-letter queue
- RACF CONTROL access to the `hlq.CONTEXT.queuename` profile if context checking is performed at the receiver
- Appropriate access to the hlq.ALTERNATE.USER.userid profiles they might need to use.
- For clients, the appropriate RACF access to the resources to be used.

APPC security
Set appropriate APPC security if you are using the LU 6.2 transmission protocol. (Use the APPCLU RACF class for example.) For information about setting up security for APPC, see the following documentation:

- *z/OS MVS Planning: APPC Management*

- *z/OS MVS Programming: Writing Servers for APPC/MVS*

  Outbound transmissions use the "SECURITY(SAME)" APPC option. As a result, the user ID of the channel initiator address space and its default profile (RACF GROUP) are flowed across the network to the receiver with an indicator that the user ID has already been verified (ALREADYV).

  If the receiving side is also z/OS, the user ID and profile are verified by APPC and the user ID is presented to the receiver channel and used as the channel user ID.

  In an environment where the queue manager is using APPC to communicate with another queue manager on the same or another z/OS system, you need to ensure that either:

  - The VTAM definition for the communicating LU specifies SETACPT(ALREADYV)
  - There is a RACF APPCLU profile for the connection between LUs that specifies CONVSEC(ALREADYV)

Changing security settings
: If the RACF access level that either the channel user ID or MCA user ID has to a destination queue is changed, this change takes effect only for new object handles (that is, new `MQOPEN` s) for the destination queue. The times when MCAs open and close queues is variable; if a channel is already running when such an access change is made, the MCA can continue to put messages on the destination queue using the existing security access of the user IDs rather than the updated security access. Stopping and restarting the channels to enforce the updated access level avoids this scenario.

Automatic restart
: If you are using the z/OS Automatic Restart Manager (ARM) to restart the channel initiator, the user ID associated with the XCFAS address space must be authorized to issue the IBM MQ START CHINIT command.

Using the Integrated Cryptographic Service Facility (ICSF)

The channel initiator can use ICSF to generate a random number when seeding the password protection algorithm to obfuscate passwords flowing over client channels if TLS is not being used. The process of generating a random number is called *entropy*.

If you have the z/OS feature installed but have not started ICSF, you see message CSQX213E and the channel initiator uses STCK for entropy.

Message CSQX213E warns you that the password protection algorithm is not as secure as it could be. However, you can continue your process; there is no other impact on runtime.

If you do not have the z/OS feature installed, the channel initiator automatically uses STCK.

Notes:

1. Using ICSF for entropy generates more random sequences than using STCK.
2. If you start ICSF you must restart the channel initiator.
3. ICSF is required for certain CipherSpecs. If you attempt to use one of these CipherSpecs and you do not have ICSF installed, you receive message CSQX629E.

z/OS Security in queue manager clusters on z/OS

Security considerations for clusters are the same for queue managers and channels that are not clustered. The channel initiator needs access to some additional system queues, and some additional commands need appropriate security set.

You can use the MCA user ID, channel authentication records, TLS, and security exits to authenticate cluster channels (as with conventional channels). The channel authentication records or security exit relating to the cluster-receiver channel must check that the remote queue manager is permitted access to the server queue manager's cluster queues. You can start to use IBM MQ cluster support without changing your existing queue access security. You must, however, allow other queue managers in the cluster to write to the SYSTEM.CLUSTER.COMMAND.QUEUE if they are to join the cluster.

IBM MQ cluster support does not provide a mechanism to limit a member of a cluster to the client role only. As a result, you must be sure that you trust any queue managers that you allow into the cluster. If any queue manager in the cluster creates a queue with a particular name, it can receive messages for that queue, regardless of whether the application putting messages to that queue intended this or not.

To restrict the membership of a cluster, take the same action that you would take to prevent queue managers connecting to receiver channels. You restrict the membership of a cluster by using channel authentication records or by writing a security exit program on the receiver channel. You can also write an exit program to prevent unauthorized queue managers from writing to the SYSTEM.CLUSTER.COMMAND.QUEUE.

Note: It is not advisable to permit applications to open the SYSTEM.CLUSTER.TRANSMIT.QUEUE directly. It is also not advisable to permit an application to open any other transmission queue directly.

If you are using resource security, consider the following points in addition to the considerations contained in "Security considerations for the channel initiator on z/OS" on page 256:

System queues
The channel initiator needs RACF ALTER access to the following system queues:

- SYSTEM.CLUSTER.COMMAND QUEUE
- SYSTEM.CLUSTER.TRANSMIT.QUEUE.

and UPDATE access to SYSTEM.CLUSTER.REPOSITORY.QUEUE

It also needs READ access to any namelists used for clustering.

Commands
Set appropriate command security (as described in Table 49 on page 217) for the cluster support commands (REFRESH and RESET CLUSTER, SUSPEND, and RESUME QMGR.

z/OS Security considerations for using IBM MQ with CICS

All the CICS versions supported by IBM MQ 9.0.0, and later, use the CICS supplied version of the adapter and bridge.

For details of security considerations, see:

- Security for the CICS-MQ adapter.
- Security for the CICS-MQ bridge.

z/OS Security considerations for using IBM MQ with IMS

Use this topic to plan your security requirements when you use IBM MQ with IMS.

Using the OPERCMDS class

If you are using RACF to protect resources in the OPERCMDS class, ensure that the userid associated with your IBM MQ queue manager address space has authority to issue the MODIFY command to any IMS system to which it can connect.

Security considerations for the IMS bridge

There are four aspects that you must consider when deciding your security requirements for the IMS bridge, these are:

- What security authorization is needed to connect IBM MQ to IMS
- How much security checking is performed on applications using the bridge to access IMS
- Which IMS resources these applications are allowed to use
- What authority is to be used for messages that are put and got by the bridge

When you define your security requirements for the IMS bridge you must consider the following:

- Messages passing across the bridge might have originated from applications on platforms that do not offer strong security features
- Messages passing across the bridge might have originated from applications that are not controlled by the same enterprise or organization

z/OS Security considerations for connecting to IMS

Grant the user ID of the IBM MQ queue manager address space access to the OTMA group.

The IMS bridge is an OTMA client. The connection to IMS operates under the user ID of the IBM MQ queue manager address space. This is normally defined as a member of the started task group. This user ID must be granted access to the OTMA group (unless the /SECURE OTMA setting is NONE).

To do this, define the following profile in the FACILITY class:

```
IMSXCF.xcfgname.mqxcfmname
```

Where `xcfgname` is the XCF group name and `mqxcfmname` is the XCF member name of IBM MQ.

You must give your IBM MQ queue manager user ID read access to this profile.

Note:

1. If you change the authorities in the FACILITY class, you must issue the RACF command SETROPTS RACLIST(FACILITY) REFRESH to activate the changes.
2. If profile hlq.NO.SUBSYS.SECURITY exists in the MQADMIN class, no user ID is passed to IMS and the connection fails unless the /SECURE OTMA setting is NONE.

z/OS Application access control for the IMS bridge

Define a RACF profile in the FACILITY class for each IMS system. Grant an appropriate level of access to the IBM MQ queue manager user ID.

For each IMS system that the IMS bridge connects to, you can define the following RACF profile in the FACILITY class to determine how much security checking is performed for each message passed to the IMS system.

```
IMSXCF.xcfgname.imsxcfmname
```

Where `xcfgname` is the XCF group name and `imsxcfmname` is the XCF member name for IMS. (You need to define a separate profile for each IMS system.)

The access level you allow for the IBM MQ queue manager user ID in this profile is returned to IBM MQ when the IMS bridge connects to IMS, and indicates the level of security that is required on subsequent transactions. For subsequent transactions, IBM MQ requests the appropriate services from RACF and, where the user ID is authorized, passes the message to IMS.

OTMA does not support the IMS /SIGN command; however, IBM MQ allows you to set the access checking for each message to enable implementation of the necessary level of control.

The following access level information can be returned:

NONE or NO PROFILE FOUND

These values indicate that maximum security is required, that is, authentication is required for every transaction. A check is made to verify that the user ID specified in the *UserIdentifier* field of the MQMD structure, and the password or PassTicket in the *Authenticator* field of the MQIIH structure are known to RACF, and are a valid combination. A UTOKEN is created with a password or PassTicket, and passed to IMS ; the UTOKEN is not cached.

Note: If profile hlq.NO.SUBSYS.SECURITY exists in the MQADMIN class, this level of security overrides whatever is defined in the profile.

READ

This value indicates that the same authentication is to be performed as for `NONE` under the following circumstances:

- The first time that a specific user ID is encountered
- When the user ID has been encountered before but the cached UTOKEN was not created with a password or PassTicket

IBM MQ requests a UTOKEN if required, and passes it to IMS.

Note: If a request to reverify security has been acted on, all cached information is lost and a UTOKEN is requested the first time each user ID is later encountered.

UPDATE

A check is made that the user ID in the *UserIdentifier* field of the MQMD structure is known to RACF.

A UTOKEN is built and passed to IMS ; the UTOKEN is cached.

CONTROL/ALTER

These values indicate that no security UTOKENs need to be provided for any user IDs for this IMS system. (You would probably only use this option for development and test systems.)

⚠ **Attention:** Note that the user ID contained in the *UserIdentifier* field of the MQMD structure is still passed for **`CONTROL/ALTER`**.

Note:

1. This access is defined when IBM MQ connects to IMS, and lasts for the duration of the connection. To change the security level, the access to the security profile must be changed and then the bridge stopped and restarted (for example, by stopping and restarting OTMA).
2. If you change the authorities in the FACILITY class, you must issue the RACF command SETROPTS RACLIST(FACILITY) REFRESH to activate the changes.
3. You can use a password or a PassTicket, but you must remember that the IMS bridge does not encrypt data. For information about using PassTickets, see "Using RACF PassTickets in the IMS header" on page 262.
4. Some of these results might be affected by security settings in IMS, using the /SECURE OTMA command.
5. Cached UTOKEN information is held for the duration defined by the INTERVAL and TIMEOUT parameters of the IBM MQ ALTER SECURITY command.
6. The RACF WARNING option has no effect on the IMSXCF.xcfgname.imsxcfmname profile. Its use does not affect the level of access granted, and no RACF WARNING messages are produced.

z/OS *Security checking on IMS*

Messages that pass across the bridge contain security information. The security checks made depend on the setting of the IMS command /SECURE OTMA.

Each IBM MQ message that passes across the bridge contains the following security information:

- A user ID contained in the *UserIdentifier* field of the MQMD structure
- The security scope contained in the *SecurityScope* field of the MQIIH structure (if the MQIIH structure is present)
- A UTOKEN (unless the IBM MQ sub system has CONTROL or ALTER access to the relevant `IMSXCF.xcfgname.imsxcfmname` profile)

The security checks made depend on the setting of the IMS command /SECURE OTMA, as follows:

/SECURE OTMA NONE

No security checks are made for the transaction.

/SECURE OTMA CHECK

The *UserIdentifier* field of the MQMD structure is passed to IMS for transaction or command authority checking.

An ACEE (Accessor Environment Element) is built in the IMS control region.

/SECURE OTMA FULL
The *UserIdentifier* field of the MQMD structure is passed to IMS for transaction or command authority checking.

An ACEE is built in the IMS dependent region as well as the IMS control region.

/SECURE OTMA PROFILE
The *UserIdentifier* field of the MQMD structure is passed to IMS for transaction or command authority checking

The *SecurityScope* field in the MQIIH structure is used to determine whether to build an ACEE in the IMS dependent region as well as the control region.

Note:

1. If you change the authorities in the TIMS or CIMS class, or the associated group classes GIMS or DIMS, you must issue the following IMS commands to activate the changes:
   - /MODIFY PREPARE RACF
   - /MODIFY COMMIT
2. If you do not use /SECURE OTMA PROFILE, any value specified in the **`SecurityScope`** field of the MQIIH structure is ignored.

z/OS *Security checking done by the IMS bridge*

Different authorities are used depending on the action being performed.

When the bridge puts or gets a message, the following authorities are used:

Getting a message from the bridge queue
No security checks are performed.

Putting an exception, or COA report message
Uses the authority of the user ID in the *UserIdentifier* field of the MQMD structure.

Putting a reply message
Uses the authority of the user ID in the *UserIdentifier* field of the MQMD structure of the original message

Putting a message to the dead-letter queue
No security checks are performed.

Note:

1. If you change the IBM MQ class profiles, you must issue the IBM MQ REFRESH SECURITY(*) command to activate the changes.
2. If you change the authority of a user, you must issue the MQSC RVERIFY SECURITY command to activate the change.

z/OS *Using RACF PassTickets in the IMS header*

You can use a PassTicket in place of a password in the IMS header.

If you want to use a PassTicket instead of a password in the IMS header (MQIIH), specify the application name against which the PassTicket is validated in the PASSTKTA attribute of the STGCLASS definition of the IMS bridge queue to which the message is to be routed.

If the PASSTKTA value is left blank, you must arrange to have a PassTicket generated. The application name in this case must be of the form MVSxxxx, where xxxx is the SMFID of the z/OS system on which the target queue manager runs.

A PassTicket is built from a user ID, the target application name, and a secret key. It is an 8-byte value containing uppercase alphabetic and numeric characters. It can be used only once, and is valid for a 20 minute period. If a PassTicket is generated by a local RACF system, RACF only checks that the profile exists and not that the user has authority against the profile. If the PassTicket was generated on a remote system, RACF validates the access of the user ID to the profile. For full information about PassTickets, see the *z/OS Security Server RACF Security Administrator's Guide*.

PassTickets in IMS headers are given to RACF by IBM MQ, not IMS.

z/OS Migrating a z/OS queue manager to mixed-case security

Follow these steps to migrate a queue manager to mixed-case security. You review the level of security product you are using and activate the new IBM MQ external security manager classes. Run the **REFRESH SECURITY** command to activate the mixed-case profiles.

Before you begin

1. Ensure all IBM MQ external security manager classes are activated.
2. Ensure your queue manager is started.

About this task

Follow these steps to convert a queue manager to mixed-case security.

Procedure

1. Copy all your existing profiles and access levels from the uppercase classes to the equivalent mixed-case external security manager class.
   a) `MQADMIN` to `MXADMIN`.
   b) `MQPROC` to `MXPROC`.
   c) `MQNLIST` to `MXNLIST`.
   d) `MQQUEUE` to `MXQUEUE`.
2. Change the value of the SCYCASE queue manager attribute to `MIXED` by issuing the following command.

```
ALTER QMGR SCYCASE(MIXED)
```

3. Activate the security profiles by issuing the following command.

```
REFRESH SECURITY(*) TYPE(CLASSES)
```

4. Test that your security profiles are working correctly.

What to do next

Review your object definitions and create new mixed-case profiles as appropriate, using the **REFRESH SECURITY** command as required to activate the profiles.

Setting up IBM MQ MQI client security

You must consider IBM MQ MQI client security, so that the client applications do not have unrestricted access to resources on the server.

When running a client application, do not run the application using a user ID that has more access rights than necessary; for example, a user in the `mqm` group or even the `mqm` user itself.

By running an application as a user with too many access rights, you run the risk of the application accessing and changing parts of the queue manager, either by accident or maliciously.

There are two aspects to security between a client application and its queue manager server: authentication and access control.

- Authentication can be used to ensure that the client application, running as a specific user, is who they say they are. By using authentication you can prevent an attacker from gaining access to your queue manager by impersonating one of your applications.

  Authentication is provided by one of two options:

  - The connection authentication feature.

    For more information on connection authentication, see "Connection authentication" on page 70.
  - Using mutual authentication within TLS.

    For more information on TLS, see "Working with SSL/TLS" on page 270.
- Access control can be used to give or remove access rights for a specific user or group of users. By running a client application with a specifically created user (or user in a specific group) you can then use access controls to ensure the application cannot access parts of your queue manager that the application is not supposed to.

  When setting up access control you must consider channel authentication rules and the MCAUSER field on a channel. Both of these features have the ability to change which user ID is being used for verifying access control rights.

  For more information on access control, see "Authorizing access to objects" on page 344.

If you have set up a client application to connect to a specific channel with a restricted ID, but the channel has an administrator ID set in its MCAUSER field then, provided the client application connects successfully, the administrator ID is used for access control checks. Therefore, the client application will have full access rights to your queue manager.

For more information on the MCAUSER attribute, see "Mapping a client user ID to an MCAUSER user ID" on page 378.

Channel authentication rules can also be used as a method for controlling access to a queue manager, by setting up specific rules and criteria for a connection to be accepted.

For more information on channel authentication rules see: "Channel authentication records" on page 51.

Specifying that only FIPS-certified CipherSpecs are used at run time on the MQI client

Create your key repositories using FIPS-compliant software, then specify that the channel must use FIPS-certified CipherSpecs.

Note: On AIX, Linux, and Windows, IBM MQ provides FIPS 140-2 compliance through the IBM Crypto for C (ICC) cryptographic module. The certificate for this module has been moved to the Historical status. Customers should view the IBM Crypto for C (ICC) certificate and be aware of any advice provided by NIST. A replacement FIPS 140-3 module is currently in progress and its status can be viewed by searching for it in the NIST CMVP modules in process list.

The IBM MQ Operator 3.2.0 and queue manager container image 9.4.0.0 onwards are based on UBI 9. FIPS 140-3 compliance is currently pending and its status can be viewed by searching for "Red Hat Enterprise Linux 9 - OpenSSL FIPS Provider" in the NIST CMVP modules in process list.

In order to be FIPS-compliant at run time, the key repositories must have been created and managed using only FIPS-compliant software such as **`runmqakm`** with the `-fips` option.

You can specify that a TLS channel must use only FIPS-certified CipherSpecs in three ways, listed in order of precedence:

1. Set the FipsRequired field in the MQSCO structure to `MQSSL_FIPS_YES`.
2. Set the environment variable **`MQSSLFIPS`** to YES.
3. Set the **`SSLFipsRequired`** attribute in the SSL stanza of the client configuration file to YES.

By default, FIPS-certified CipherSpecs is not required.

These values have the same meanings as the equivalent parameter values on **ALTER QMGR SSLFIPS** (see **ALTER QMGR** (alter queue manager settings)). If the client process currently has no active TLS connections, and a FipsRequired value is validly specified on an SSL MQCONNX, all subsequent TLS connections associated with this process must use only the CipherSpecs associated with this value. This applies until this and all other TLS connections have stopped, at which stage a subsequent MQCONNX can provide a new value for FipsRequired.

If cryptographic hardware is present, the cryptographic modules used by IBM MQ can be configured to be those modules provided by the hardware product, and these might be FIPS-certified to a particular level. The configurable modules and whether they are FIPS-certified depends on the hardware product in use.

Where possible, if FIPS-only CipherSpecs is configured then the MQI client rejects connections which specify a non-FIPS CipherSpec with MQRC_SSL_INITIALIZATION_ERROR. IBM MQ does not guarantee to reject all such connections and it is your responsibility to determine whether your IBM MQ configuration is FIPS-compliant.

Related concepts

"Federal Information Processing Standards (FIPS) for AIX, Linux, and Windows" on page 35
When cryptography is required on an SSL/TLS channel on AIX, Linux, and Windows systems, IBM MQ uses a cryptography package called IBM Crypto for C (ICC). On the AIX, Linux, and Windows platforms, the ICC software has passed the Federal Information Processing Standards (FIPS) Cryptomodule Validation Program of the US National Institute of Standards and Technology, at level 140-2.

AIX Running TLS client applications with multiple installations of GSKit 8.0 on AIX

TLS client applications on AIX might experience MQRC_CHANNEL_CONFIG_ERROR and error AMQ6175 when running on AIX systems with multiple IBM Global Security Kit (GSKit) 8.0 installations.

When running client applications on an AIX system with multiple GSKit 8.0 installations, the client connect calls can return MQRC_CHANNEL_CONFIG_ERROR when using TLS. The /var/mqm/errors logs record error AMQ6175 and AMQ9220 for the failing client application, for example:

```
09/08/11 11:16:13 - Process(24412.1) User(user) Program(example)
Host(machine.example.ibm.com) Installation(Installation1)
VRMF(7.1.0.0)
AMQ6175: The system could not dynamically load the shared library
'/usr/mqm/gskit8/lib64/libgsk8ssl_64.so'. The system returned
error number '8' and error message 'Symbol resolution failed
for /usr/mqm/gskit8/lib64/libgsk8ssl_64.so because:
Symbol VALUE_EC_NamedCurve_secp256r1__9GSKASNOID (number 16) is not
exported from dependent module /db2data/db2inst1/sqllib/lib64/libgsk8cms_64.so.
Symbol VALUE_EC_NamedCurve_secp384r1__9GSKASNOID (number 17) is not exported from
dependent module /db2data/db2inst1/sqllib/lib64/libgsk8cms_64.so.
Symbol VALUE_EC_NamedCurve_secp521r1__9GSKASNOID (number 18) is not exported from
dependent module /db2data/db2inst1/sqllib/lib64/libgsk8cms_64.so.
Symbol VALUE_EC_ecPublicKey__9GSKASNOID (number 19) is not exported from dependent
module /db2data/db2inst1/sqllib/lib64/libgsk8cms_64.so.
Symbol VALUE_EC_ecdsa_with_SHA1__9GSKASNOID (number 20) is not exported from
dependent module /db2data/db2inst1/sqllib/lib64/libgsk8cms_64.so.
Symbol VALUE_EC_ecdsa__9GSKASNOID (number 21) is not exported from dependent
module /db2data/db2inst1/sqllib/lib64/libgsk8cms_64.so.'.

EXPLANATION:
This message applies to AIX systems. The shared library
'/usr/mqm/gskit8/lib64/libgsk8ssl_64.so' failed
to load correctly due to a problem with the library.
ACTION:
Check the file access permissions and that the file has not been corrupted.
----- amqxufnx.c : 1284 --------------------------------------------------------
09/08/11 11:16:13 - Process(24412.1) User(user) Program(example)
Host(machine.example.ibm.com) Installation(Installation1)
VRMF(7.1.0.0)
AMQ9220: The GSKit communications program could not be loaded.

EXPLANATION:
The attempt to load the GSKit library or procedure
'/usr/mqm/gskit8/lib64/libgsk8ssl_64.so' failed with error code
536895861.
```

```
ACTION:
Either the library must be installed on the system or the environment changed
to allow the program to locate it.
----- amqcgska.c : 836 --------------------------------------------------------
```

A common cause of this error is that the setting of the `LIBPATH` or `LD_LIBRARY_PATH` environment variable has caused the IBM MQ client to load a mixed set of libraries from two different GSKit 8.0 installations. Executing an IBM MQ client application in a Db2 environment can cause this error.

To avoid this error, include the IBM MQ library directories at the front of the library path so that the IBM MQ libraries take precedence. This can be achieved using the **`setmqenv`** command with the **`-k`** parameter, for example:

```

. /usr/mqm/bin/setmqenv -s -k
```

For more information about the use of the **`setmqenv`** command, refer to setmqenv (set IBM MQ environment)

Configuring TLS channels with MQSC

To configure TLS channels, use the **`runmqsc`** and the ALTER CHANNEL commands. You can optionally configure a channel to accept only certificates with attributes in the distinguished name of the owner that match given values. You can also optionally configure a queue manager channel so that the queue manager refuses the connection if the initiating party does not send its own personal certificate.

About this task

To configure channels in IBM MQ Explorer, see Configuring TLS channels with IBM MQ Explorer.

To configure channels using **`runmqsc`**, complete the following steps.

Procedure

1. Invoke the **`runmqsc`** command connecting to the target queue manager.
2. Identify the channel you want to enable for TLS.

   Note both the channel name and channel type.
3. Use the ALTER CHANNEL command to alter various properties of an IBM MQ channel.

   You provide the channel name and channel type in addition to the command. For example, to alter a sender channel called MQ.TEST run the following command:

   ```
   ALTER CHANNEL('MQ.TEST') CHLTYPE(SDR)
   ```

   There are various channel attributes related to TLS that you can adjust on IBM MQ channel definitions.

What to do next

Setting message security

TLS-enabled messaging offers two methods of ensuring message security:

- Encryption ensures that if the message is intercepted, it is unreadable.
- Hash functions ensure that if the message is altered, this is detected.

The combination of these methods is called the cipher specification, or CipherSpec. The same CipherSpec must be set for both ends of a channel, otherwise TLS-enabled messaging fails. For more information, see "Securing IBM MQ" on page 7.

To alter an IBM MQ channel enable TLS, specify a value in the SSLCIPH attribute. This attribute must be set to a valid CipherSpec for the queue platform of the queue manager from the list "Enabling CipherSpecs" on page 411.

To alter an IBM MQ channel to disable TLS, set SSLCIPH to a blank value. For example:

```
ALTER CHANNEL(‘MQ.TEST’) CHLTYPE(SDR) SSLCIPH(ANY_TLS12_OR_HIGHER)
```

Note: You must encase the channel name within single quotes to ensure the character case is maintained. Without single quotes, IBM MQ transforms the string to be all uppercase.

Filtering certificates on their owner's name

Certificates contain the distinguished name of the owner of the certificate. You can optionally configure the channel to accept only certificates with attributes in the distinguished name of the owner that match given values.

The attribute names that IBM MQ can filter are listed in the following table:

Attribute names	**Meaning**
SERIALNUMBER	Certificate serial number
MAIL	Email address
Deprecated E	Email address (Deprecated in preference to MAIL)
UID or USERID	User identifier
CN	Common Name
T	Title
OU	Organizational Unit name
DC	Domain component
O	Organization name
STREET	Street / First line of address
L	Locality name
ST (or SP or S)	State or Province name
PC	Postal code / zip code
C	Country
UNSTRUCTUREDNAME	Host name
UNSTRUCTUREDADDRESS	IP address
DNQ	Distinguished name qualifier

You can use the wildcard character (*) at the beginning or the end of the attribute value in place of any number of characters. For example, to accept only certificates from any person with a name ending with `Smith` working for `IBM` in `GB`, type:

```
CN=*Smith, O=IBM, C=GB
```

For example:

```
ALTER CHANNEL(‘MQ.TEST’) CHLTYPE(SDR) SSLPEER(‘CN=*Smith, O=IBM, C=GB’)
```

Note: You must encase the SSLPEER string within single quotes to ensure the character case is maintained. Without single quotes, IBM MQ transforms the string to be all uppercase.

Authenticating parties initiating connections to a queue manager

When another party initiates a TLS-enabled connection to a queue manager, the queue manager must send its personal certificate to the initiating party as proof of identity. You can also optionally configure the queue manager channel so that the queue manager refuses the connection if the initiating party does not send its own personal certificate.

To do this, set the SSLCAUTH attribute. This attribute is a Boolean attribute and can have the values OPTIONAL or REQUIRED:

- OPTIONAL authenticates the certificate of a connecting client if one is provided but does not require a client to send one. A client is rejected if it sends a certificate that is not valid.
- REQUIRED rejects any connecting clients that do not provide a valid TLS certificate

For example:

```
ALTER CHANNEL('MQ.TEST') CHLTYPE(SDR) SSLCAUTH(REQUIRED)
```

IBM i Setting up communications for SSL or TLS on IBM i

Secure communications that use the SSL or TLS cryptographic security protocols involve setting up the communication channels and managing the digital certificates that you will use for authentication.

To set up your SSL or TLS installation you must define your channels to use SSL or TLS. You must also create and manage your digital certificates. On some operating systems, you can perform the tests with self-signed certificates. However, on IBM i, you must use personal certificates signed by a local CA.

For full information about creating and managing certificates, see "Working with SSL/TLS on IBM i" on page 270.

This collection of topics introduces some of the tasks involved in setting up SSL or TLS communications, and provides step-by-step guidance on completing those tasks

You might also want to test SSL or TLS client authentication, which are optional parts of the SSL and TLS protocols. During the SSL or TLS handshake, the SSL or TLS client always obtains and validates a digital certificate from the server. With the IBM MQ implementation, the SSL or TLS server always requests a certificate from the client.

On IBM i, the SSL or TLS client sends a certificate only if it has one labeled in the correct IBM MQ format:

- For a queue manager, `ibmwebspheremq` followed by the name of your queue manager changed to lowercase. For example, for `QM1`, `ibmwebspheremqqm1`.
- For an IBM MQ C Client for IBM i, `ibmwebspheremq` followed by your logon user ID changed to lowercase, for example `ibmwebspheremqmyuserid`.

IBM MQ uses the `ibmwebspheremq` prefix on a label to avoid confusion with certificates for other products. Ensure that you specify the entire certificate label in lowercase.

The SSL or TLS server always validates the client certificate if one is sent. If the SSL or TLS client does not send a certificate, authentication fails only if the end of the channel acting as the SSL or TLS server is defined with either the SSLCAUTH parameter set to REQUIRED or an SSLPEER parameter value set. For more information, see Connecting two queue managers using SSL or TLS.

ALW Setting up communications for SSL or TLS on AIX, Linux, and Windows

Secure communications that use the SSL or TLS cryptographic security protocols involve setting up the communication channels and managing the digital certificates that you will use for authentication.

To set up your SSL or TLS installation you must define your channels to use SSL or TLS. You must also create and manage your digital certificates. On AIX, Linux, and Windows systems, you can perform the tests with self-signed certificates.

Attention: It is not possible to use a mixture of Elliptic Curve-signed certificates and RSA-signed certificates on queue managers that you want to join together using TLS enabled channels.

Queue managers using TLS enabled channels must all use RSA-signed certificates, or all use EC-signed certificates, not a mixture of both.

See "Digital certificates and CipherSpec compatibility in IBM MQ" on page 47 for more information.

Self-signed certificates cannot be revoked, which could allow an attacker to spoof an identity after a private key has been compromised. CAs can revoke a compromised certificate, which prevents its further use. CA-signed certificates are therefore safer to use in a production environment, though self-signed certificates are more convenient for a test system.

For full information about creating and managing certificates, see "Working with SSL/TLS on AIX, Linux, and Windows" on page 287.

This collection of topics introduces some of the tasks involved in setting up SSL communications, and provides step-by-step guidance on completing those tasks.

You might also want to test SSL or TLS client authentication, which are an optional part of the protocols. During the SSL or TLS handshake, the SSL or TLS client always obtains and validates a digital certificate from the server. With the IBM MQ implementation, the SSL or TLS server always requests a certificate from the client.

On AIX, Linux, and Windows, the SSL or TLS client sends a certificate only if it has one labeled in the correct IBM MQ format:

- For a queue manager, the format is `ibmwebspheremq` followed by the name of your queue manager changed to lowercase. For example, for `QM1`, `ibmwebspheremqqm1`
- For an IBM MQ client, `ibmwebspheremq` followed by your logon user ID changed to lowercase, for example `ibmwebspheremqmyuserid`.

IBM MQ uses the `ibmwebspheremq` prefix on a label to avoid confusion with certificates for other products. Ensure that you specify the entire certificate label in lowercase.

The SSL or TLS server always validates the client certificate if one is sent. If the client does not send a certificate, authentication fails only if the end of the channel acting as the SSL or TLS server is defined with either the SSLCAUTH parameter set to REQUIRED or an SSLPEER parameter value set. For more information, see Connecting two queue managers using SSL or TLS.

z/OS Setting up communications for SSL or TLS on z/OS

Secure communications that use the SSL or TLS cryptographic security protocols involve setting up the communication channels and managing the digital certificates that you will use for authentication.

To set up your SSL or TLS installation you must define your channels to use SSL or TLS. You must also create and manage your digital certificates. On z/OS you can perform the tests with self-signed certificates, or with personal certificates signed by a local certificate authority (CA).

Self-signed certificates cannot be revoked, which could allow an attacker to spoof an identity after a private key has been compromised. CAs can revoke a compromised certificate, which prevents its further use. CA-signed certificates are therefore safer to use in a production environment, though self-signed certificates are more convenient for a test system.

For full information about creating and managing certificates, see "Working with SSL/TLS on z/OS" on page 301.

See the CERTLABL and CERTQSGL parameters of the ALTER QMGR command and the CERLABL parameter of the DEFINE CHANNEL command for more information.

The order of precedence is:

- Channel CERTLABL parameter
- QMGR CERTQSGL parameter if the channel is shared.

  For a sender channel, that means the transmission queue (XMITQ) is shared. For a receiver channel, that means the channel started through the shared listener, that is the listener with INDISP(GROUP).

- QMGR CERTLABL
- The default label of `ibmWebSphereMQ` followed by the name of the queue sharing group for shared channels, or the name of the queue manager.

This collection of topics introduces some of the tasks involved in setting up SSL or TLS communications, and provides step-by-step guidance on completing those tasks.

You might also want to test SSL or TLS client authentication, which are an optional part of the protocols. During the SSL or TLS handshake, the SSL or TLS client always obtains and validates a digital certificate from the server. With the IBM MQ implementation, the SSL or TLS server always requests a certificate from the client.

If the channel is shared, the channel first tries to find a certificate for the queue sharing group. If it does not find a certificate for a queue sharing group, it tries to find a certificate for the queue manager.

On z/OS, IBM MQ uses the `ibmWebSphereMQ` prefix on a label to avoid confusion with certificates for other products.

The SSL or TLS server always validates the client certificate if one is sent. If the SSL or TLS client does not send a certificate, authentication fails only if the end of the channel acting as the SSL or TLS server is defined with either the SSLCAUTH parameter set to REQUIRED or an SSLPEER parameter value set. For more information, see Connecting two queue managers using SSL or TLS.

Working with SSL/TLS

These topics give instructions for performing single tasks related to using TLS with IBM MQ.

Many of them are used as steps in the higher-level tasks described in the following sections:

- "Identifying and authenticating users" on page 313
- "Authorizing access to objects" on page 344
- "Confidentiality of messages" on page 410
- "Data integrity of messages" on page 465
- "Keeping clusters secure" on page 466

IBM i Working with SSL/TLS on IBM i

This collection of topics gives instructions for individual tasks working with Transport Layer Security (TLS) in IBM MQ for IBM i.

For IBM i the TLS support is integral to the operating system. Ensure that you have installed the prerequisites listed in Hardware and software requirements on IBM i.

On IBM i, you manage keys and digital certificates with the Digital Certificate Manager (DCM) tool.

IBM i *Accessing DCM*

Follow these instructions to access the DCM interface.

About this task

Perform the following steps in a web browser that supports frames.

Procedure

1. Go to either `http://`*`machine`*`.domain:2001` or `https://`*`machine`*`.domain:2010`, where *machine* is the name of your computer.
2. Type a valid user profile and password when requested to.

   Ensure that your user profile has *ALLOBJ and *SECADM special authorities to enable you to create new certificate stores. If you do not have the special authorities, you can only manage your personal certificates or view the object signatures for the objects for which you are authorized. If you are authorized to use an object signing application, you can also sign objects from DCM.

3. On the Internet Configurations page, click **Digital Certificate Manager**.

   The Digital Certificate Manager page is displayed.

IBM i *Assigning a certificate to a queue manager on IBM i*

Use DCM to assign a certificate to a queue manager.

Use traditional IBM i digital certificate management to assign a certificate to a queue manager. This means that you can specify that a queue manager uses the system certificate store, and that the queue manager is registered for use as an application with Digital Certificate Manager. To do this, change the value of the queue manager **SSLKEYR** attribute to `*SYSTEM`.

When the **SSLKEYR** parameter is changed to `*SYSTEM`, IBM MQ registers the queue manager as a server application with a unique application label of QIBM_WEBSPHERE_MQ_QMGRNAME and a label with a description of Qmgrname (WMQ). Note that channel **CERTLABL** attributes are not used if you use the *SYSTEM certificate store. The queue manager then appears as a server application in Digital Certificate Manager, and you can assign to this application any server or client certificate in the system store.

Because the queue manager is registered as an application, advanced features of DCM such as defining CA trust lists can be carried out.

If the **SSLKEYR** parameter is changed to a value other than `*SYSTEM`, IBM MQ deregisters the queue manager as an application with Digital Certificate Manager. If a queue manager is deleted, it is also deregistered from DCM. A user with sufficient *SECADM authority can also manually add or remove applications from DCM.

IBM i *Setting up a key repository on IBM i*

A key repository must be set up at both ends of the connection. The default certificate stores can be used or you can create your own.

A TLS connection requires a *key repository* at each end of the connection. Each queue manager and IBM MQ MQI client must have access to a key repository. If you want to access the key repository using a file name and password (that is, not using the *SYSTEM option) ensure that the QMQM user profile has the following authorities:

- Execute authority for the directory containing the key repository
- Read authority for the file containing the key repository

See "The SSL/TLS key repository" on page 25 for more information. Note that channel **CERTLABL** attributes are not used if you use the *SYSTEM certificate store.

On IBM i, digital certificates are stored in a certificate store that is managed with DCM. These digital certificates have labels, which associate a certificate with a queue manager or an IBM MQ MQI client. TLS uses the certificates for authentication purposes.

The label is either the value of the **CERTLABL** attribute, if it is set, or the default `ibmwebspheremq` with the name of the queue manager or IBM MQ MQI client user logon ID appended, all in lowercase. See Digital certificate labels for details.

The queue manager or IBM MQ MQI client certificate store name comprises a path and stem name. The default path is `/QIBM/UserData/ICSS/Cert/Server/` and the default stem name is `Default`. On IBM i, the default certificate store, `/QIBM/UserData/ICSS/Cert/Server/Default.kdb`, is also known as `*SYSTEM`. Optionally, you can define your own path and stem name.

If you define your own path or file name, set the permissions to the file to tightly control access to it.

"Changing the key repository location for a queue manager on IBM i" on page 274 tells you about specifying the certificate store name. You can specify the certificate store name either before or after creating the certificate store.

Note: The operations you can perform with DCM might be limited by the authority of your user profile. For example, you require *ALLOBJ and *SECADM authorities to create a CA certificate.

IBM i *Encrypting key repository passwords on IBM i*

Several IBM MQ components need access to a key repository that contains digital certificates or symmetric keys. A key repository is secured with a password as it contains sensitive information. The key repository password must be stored in a location where IBM MQ can read it when the key repository is accessed. The password must also be encrypted to reduce the likelihood of unauthorized access to the key repository.

The following IBM MQ components and features support two different methods to store key repository passwords:

- The queue manager TLS key repository.
- IBM MQ MQI clients that use TLS.

Key repository passwords for use by these components are protected using the IBM MQ password protection system. The mechanism for providing a password and encrypting it varies slightly depending on component:

The queue manager TLS key repository

The password is encrypted when the **SSLKEYRPWD** queue manager attribute is set using the CHGMQM (Change Message Queue Manager) command.

The password is encrypted with the AES-128 algorithm. The details of this algorithm are publicly known and it is considered secure.

The password is stored in a stash file in a proprietary format that is not understood by other software that might access the key repository.

A password that is encrypted by one IBM MQ component cannot be used by a different IBM MQ component.

A unique encryption key can be provided when the key repository password is encrypted. A unique encryption key prevents anyone who does not have access to the encryption key from being able to decrypt the password. You provide this key through the **INITKEY** queue manager attribute, which must be set before you provide a password to be encrypted.

For more information about the IBM MQ password protection system, see "Protecting passwords in IBM MQ component configuration files" on page 551.

IBM MQ MQI clients that use TLS

The "IBM MQ SSL Client utility (amqrsslc) for IBM i" on page 285 can store the key repository password in a stash file. See also Administering using MQSC commands on IBM i.

The password is encrypted with the AES-128 algorithm. The details of this algorithm are publicly known and it is considered secure.

The password is stored in a stash file in a proprietary format that is not understood by other software that might access the key repository.

A unique encryption key can be provided when the key repository password is encrypted. A unique encryption key prevents anyone who does not have access to the encryption key from being able to decrypt the password. You provide this key through the **-sf** parameter.

The encrypted password is stored in a stash file in the same directory as the key repository file.

IBM MQ MQI clients also support passwords provided through other mechanisms. See "Supplying the key repository password for an IBM MQ MQI client on IBM i" on page 276.

Regardless of the method that you choose to encrypt the key repository password, ensure that you are aware of the limitations of encrypting stored passwords. See "The limits to protection through password encryption" on page 558.

Related concepts

“Supplying the key repository password for a queue manager on IBM i” on page 275
Because the key repository contains sensitive information, it is secured with a password. To be able to access the key repository contents to perform TLS operations, IBM MQ must be able to retrieve the key repository password.

“Supplying the key repository password for an IBM MQ MQI client on IBM i” on page 276
Because the key repository contains sensitive information, it is secured with a password. To be able to access the key repository contents to perform TLS operations, IBM MQ must be able to retrieve the key repository password.

“Working with SSL/TLS on IBM i” on page 270
This collection of topics gives instructions for individual tasks working with Transport Layer Security (TLS) in IBM MQ for IBM i.

IBM i *Creating a certificate store on IBM i*
If you do not want to use the default certificate store, follow this procedure to create your own.

About this task

Create a new certificate store only if you do not want to use the IBM i default certificate store.

To specify that the IBM i system certificate store is to be used, change the value of the queue manager's SSLKEYR attribute to *SYSTEM. This value indicates that the queue manager uses the system certificate store, and the queue manager is registered for use as an application with Digital Certificate Manager (DCM).

Procedure

1. Access the DCM interface, as described in “Accessing DCM” on page 270
2. In the navigation panel, click **Create New Certificate Store**.

   The Create New Certificate Store page is displayed in the task frame.
3. In the task frame, select **Other System Certificate Store** and click **Continue**.

   The Create a Certificate in New Certificate Store page is displayed in the task frame.
4. Select **No - Do not create a certificate in the certificate store** and click **Continue**.

   The Certificate Store Name and Password page is displayed in the task frame.
5. In the **Certificate store path and filename** field, type an IFS path and file name, for example `/QIBM/UserData/mqm/qmgrs/qm1/key.kdb`
6. Type a password in the **Password** field and type it again in the **Confirm Password** field. Click **Continue**.

   Make a note of the password (which is case sensitive) because you need it when you stash the repository key.
7. To exit from DCM, close your browser window.

What to do next

When you have created the certificate store using DCM, ensure you stash the password, as described in “Stashing the certificate store password on IBM i systems” on page 273

Related tasks

“Importing a certificate into a key repository on IBM i” on page 283
Follow this procedure to import a certificate.

IBM i *Stashing the certificate store password on IBM i systems*
Stash the certificate store password by using CL commands.

The following instructions apply to stashing the certificate store password on IBM i for a queue manager. Alternatively, for an IBM MQ MQI client, if you are not using the *SYSTEM certificate store (that is, the

MQSSLKEYR environment is set to a value other than *SYSTEM), follow the procedure described in the "Stash the certificate store password" on page 286 section of "IBM MQ SSL Client utility (amqrsslc) for IBM i" on page 285.

If you have specified that the *SYSTEM certificate store is to be used (by changing the value of the SSLKEYR attribute of the queue manager to *SYSTEM) you must not follow these steps.

When you have created the certificate store using DCM, use the following commands to stash the password:

```
STRMQM MQMNAME('queue_manager_name')
CHGMQM MQMNAME('queue_manager_name') SSLKEYRPWD('password')
```

The password is case sensitive. It must be entered in single quotation marks exactly as you entered it in step 6 of "Creating a certificate store on IBM i" on page 273.

Note: If you are not using the default system certificate store, and you do not stash the password, attempts to start TLS channels fail because they cannot obtain the password required to access the certificate store.

Password protection

When a key repository password is specified, IBM MQ encrypts the password using the IBM MQ Password Protection system. To encrypt the password an initial key is used; if this is not supplied to the queue manager, a default key is used instead.

Prior to supplying the key repository password, you should set a unique initial key for the queue manager. You can do this by using the **INITKEY** attribute of the **ALTER QMGR** MQSC command:

```
ALTER QMGR INITKEY('value')
```

IBM i *Locating the key repository for a queue manager on IBM i*

Use this procedure to obtain the location of your queue manager's certificate store.

Procedure

1. Display your queue manager's attributes, using the following command:

   ```
   DSPMQM MQMNAME('queue manager name')
   ```

2. Examine the command output for the path and stem name of the certificate store.

   For example: `/QIBM/UserData/ICSS/Cert/Server/Default`, where `/QIBM/UserData/ICSS/Cert/Server` is the path and `Default` is the stem name.

IBM i *Changing the key repository location for a queue manager on IBM i*

Change the location of your queue manager's certificate store using either CHGMQM or ALTER QMGR.

Procedure

Use either the CHGMQM command or the ALTER QMGR MQSC command to set your queue manager's key repository attribute.

a) Using CHGMQM: `CHGMQM MQMNAME('qm1') SSLKEYR('/QIBM/UserData/ICSS/Cert/Server/MyKey.kdb')`

b) Using ALTER QMGR: `ALTER QMGR SSLKEYR('/QIBM/UserData/ICSS/Cert/Server/MyKey.kdb')`

In either case, the certificate store has the fully qualified file name: `/QIBM/UserData/ICSS/Cert/Server/MyKey.kdb`

What to do next

When you change the location of a queue manager's certificate store, certificates are not transferred from the old location. If the CA certificates preinstalled when you create the certificate store are insufficient, you must populate the new certificate store with certificates, as described in "Importing a certificate into a key repository on IBM i" on page 283. You must also stash the password for the new location, as described in "Stashing the certificate store password on IBM i systems" on page 273.

IBM i *Supplying the key repository password for a queue manager on IBM i*

Because the key repository contains sensitive information, it is secured with a password. To be able to access the key repository contents to perform TLS operations, IBM MQ must be able to retrieve the key repository password.

IBM MQ provides a mechanism to supply the key repository password to a queue manager:

- The **SSLKEYRPWD** parameter on the **CHGMQM** command

The key repository password is encrypted by using the IBM MQ password protection system. For more information about the methods of protecting the key repository password, see "Encrypting key repository passwords on IBM i" on page 272.

See also Administering using MQSC commands on IBM i.

The SSLKEYRPWD attribute

To supply a key repository password directly to the queue manager, run the following **CHGMQM** command, replacing *queue_manager* with your queue manager name, and *password* with your key repository password.

```
CHGMQM MQMNAME('queue_manager') SSLKEYRPWD('password')
```

Attention: Ensure that you surround the queue manager name and password with single quotation marks, otherwise IBM MQ converts the characters to uppercase.

When a key repository password is specified by using this method, the password is encrypted by using the IBM MQ password protection system before it is stored.

An encryption key, which is known as the initial key, is used to encrypt the password. Set the queue manager to use a unique initial key to securely protect the password. If you do not supply an initial key, the default key is used.

Ensure that the queue manager is configured with a unique initial key before you set the key repository password. You can modify the initial key by using the **INITKEY** attribute on the **ALTER QMGR** command. For example:

```
ALTER QMGR INITKEY('mykey')
```

Warning: If you modify the initial key after setting the key repository password, the key repository password is not encrypted with the new initial key. If you change the initial key, you must also reset the key repository password. Otherwise IBM MQ cannot decrypt the key repository password, and therefore cannot access the key repository.

For more information about the **SSLKEYRPWD** attribute, see The **SSLKEYRPWD** parameter on the **CHGMQM** command.

Related concepts

"Encrypting key repository passwords on IBM i" on page 272
Several IBM MQ components need access to a key repository that contains digital certificates or symmetric keys. A key repository is secured with a password as it contains sensitive information. The key repository password must be stored in a location where IBM MQ can read it when the key repository is accessed. The password must also be encrypted to reduce the likelihood of unauthorized access to the key repository.

"Supplying the key repository password for an IBM MQ MQI client on IBM i" on page 276

Because the key repository contains sensitive information, it is secured with a password. To be able to access the key repository contents to perform TLS operations, IBM MQ must be able to retrieve the key repository password.

IBM i *Supplying the key repository password for an IBM MQ MQI client on IBM i*

Because the key repository contains sensitive information, it is secured with a password. To be able to access the key repository contents to perform TLS operations, IBM MQ must be able to retrieve the key repository password.

IBM MQ provides four mechanisms to supply the key repository password to a IBM MQ MQI client:

- "The KeyRepoPassword fields of MQSCO " on page 276
- "The MQKEYRPWD environment variable" on page 276
- "The SSLKeyRepositoryPassword attribute of the client configuration file" on page 277
- "The key repository stash file" on page 277

If you do not use a key repository stash file, you can supply the key repository password as a plain text string, or a string that is encrypted by using the IBM MQ password protection system. For more information about the methods of protecting the key repository password, see "Encrypting key repository passwords on IBM i" on page 272.

The KeyRepoPassword fields of MQSCO

To supply a key repository password by using the MQSCO structure, you must use a combination of the following three variable string fields:

KeyRepoPasswordLength
The length of the password.

KeyRepoPasswordPtr
A pointer to the location in memory that contains the password.

KeyRepoPasswordOffset
The location of the password in memory, represented as number of bytes from the start of the MQSCO structure.

Note: You can supply only one of **KeyRepoPasswordPtr** or **KeyRepoPasswordOffset**.

For example:

```
char * pwd = "passw0rd";
MQSCO  SslConnOptions = {MQSCO_DEFAULT};

SslConnOptions.KeyRepoPasswordPtr = pwd;
SslConnOptions.KeyRepoPasswordLength = (MQLONG)strlen(SslConnOptions.KeyRepoPasswordPtr);
SslConnOptions.Version = MQSCO_VERSION_6;
```

Attention: If you supply the password by using this method, encrypt the password before it is supplied to the IBM MQ client application. For more information, see "Encrypting the key repository password" on page 277.

For more information about the MQCSO structure, see MQSCO - SSL/TLS configuration options.

The MQKEYRPWD environment variable

If a key repository password is not supplied to the client by using the MQSCO structure, you can specify the key repository password by using the **MQKEYRPWD** environment variable. For example:

```
export MQKEYRPWD=passw0rd
```

or

```
set MQKEYRPWD=passw0rd
```

where *passw0rd* is your password.

⚠ **Attention:** If you supply the password by using this method, encrypt the password before you set the value of the environment variable. For more information, see "Encrypting the key repository password" on page 277.

The SSLKeyRepositoryPassword attribute of the client configuration file

If a key repository password is not supplied to the client by using one of the other methods, you can specify the key repository password by using the **SSLKeyRepositoryPassword** attribute in the **SSL** stanza of the client configuration file. For example:

```
SSL:
    SSLKeyRepositoryPassword=passw0rd
```

⚠ **Attention:** If you supply the password by using this method, encrypt the password before setting the value of the **SSLKeyRepositoryPassword** attribute. For more information, see "Encrypting the key repository password" on page 277.

Ford more information about the SSL stanza of the client configuration file, see SSL stanza of the client configuration file.

The key repository stash file

If the key repository password is not supplied to the client by using one of the other methods, IBM MQ assumes that a stash file exists in the same directory as the key repository. The stash file has the same stem name as the key repository, but has the `.sth` extension.

A key repository stash file is created using the **amqrsslc** command line tool. To create the stash file, run the following command:

```
CALL PGM(QMQM/AMQRSSLC) PARM('-s' '/Path/Of/KeyDatabase/MyKey')
```

This command prompts you for the password to encrypt. The password is encrypted by the IBM MQ password protection system, with a default encryption key unless one is provided using the **-sf** parameter.

For more information, see "IBM MQ SSL Client utility (amqrsslc) for IBM i" on page 285 and "Encrypting the key repository password" on page 277.

Encrypting the key repository password

If you supply the key repository password by using any method other than a stash file, encrypt the password by using the IBM MQ password protection system. To encrypt the password, run the **runmqicred** command. Enter the key repository password when prompted. The command outputs the encrypted password. The encrypted password can be supplied to the IBM MQ MQI client instead of the plain text password by using any of the methods described.

An encryption key, which is known as the initial key, is used to encrypt the password. When you encrypt the password, use a unique initial key to securely protect the password. To supply your own initial key, use the **-sf** parameter to the **runmqicred** command. If you do not supply an initial key, the default key is used.

For more information, see runmqicred (protect IBM MQ client passwords).

If you supply your own initial key when the key repository password is encrypted, and provide the encrypted password to the IBM MQ MQI client, you must also ensure that you supply the same initial key to the IBM MQ MQI client. For more information about how to provide the initial key to an IBM MQ MQI client, see "Supplying an initial key for an IBM MQ MQI client on IBM i" on page 278.

Related concepts

"Encrypting key repository passwords on IBM i" on page 272

Several IBM MQ components need access to a key repository that contains digital certificates or symmetric keys. A key repository is secured with a password as it contains sensitive information. The key repository password must be stored in a location where IBM MQ can read it when the key repository is accessed. The password must also be encrypted to reduce the likelihood of unauthorized access to the key repository.

"Supplying the key repository password for a queue manager on IBM i" on page 275
Because the key repository contains sensitive information, it is secured with a password. To be able to access the key repository contents to perform TLS operations, IBM MQ must be able to retrieve the key repository password.

IBM i *Supplying an initial key for an IBM MQ MQI client on IBM i*
If you supply variables to an IBM MQ MQI client that have been encrypted using the IBM MQ Password Protection System, you might need to supply the corresponding initial key that was used to encrypt the value.

If you did not specify an initial key when encrypting the value, you do not need to provide any initial key value to the IBM MQ client. However, if you used a unique initial key you can provide the initial key to the IBM MQ client using the following methods:

- "Supplying the initial key using the MQCSP structure" on page 278
- "Supplying the initial key using the MQS_MQI_KEYFILE environment variable" on page 278
- "Supplying the initial key using the client configuration file" on page 279

Supplying the initial key using the MQCSP structure

To supply the initial key using the MQCSP structure, you must use a combination of the following three variable string fields:

InitialKeyLength
The length of the initial key

InitialKeyPtr
A pointer to the location in memory containing the initial key

InitialKeyOffset
The location of the initial key in memory, represented as number of bytes from the start of the MQCSP structure.

Note: You can supply only one of **InitialKeyPtr** or **InitialKeyOffset**.

For example:

```
char * initialKey = "myInitialKey";
MQCSP  cspOptions = {MQCSP_DEFAULT};


cspOptions.InitialKeyPtr = initialKey;
cspOptions.InitialKeyLength = (MQLONG)strlen(cspOptions.InitialKeyPtr);
cspOptions.Version = MQCSP_VERSION_2;
```

Supplying the initial key using the MQS_MQI_KEYFILE environment variable

If an initial key is not supplied to the client using the MQCSP structure, IBM MQ checks the *MQS_MQI_KEYFILE* environment variable. You should set this environment variable to the location of a file containing a single line of text, consisting of the initial key you want to use.

For example, if a file called mykey.key exists in the root directory, and contains the initial key, you should set the environment variable as follows:

```
export MQS_MQI_KEYFILE=/mykey.key
```

or

```
set MQS_MQI_KEYFILE=C:\mykey.key
```

Supplying the initial key using the client configuration file

If an initial key is not supplied to the client using a previous mechanism, IBM MQ checks the **`MQIInitialKeyFile`** attribute of the Security stanza of the `mqclient.ini` file. You should set this attribute to the location of a file containing a single line of text, consisting of the initial key you want to use.

For example, if a file called `mykey.key` exists in the root directory, and contains the initial key, the client configuration file should contain the following:

```

Security:
    MQIInitialKeyFile=/mykey.key
```

Related concepts

"Encrypting key repository passwords on IBM i" on page 272
Several IBM MQ components need access to a key repository that contains digital certificates or symmetric keys. A key repository is secured with a password as it contains sensitive information. The key repository password must be stored in a location where IBM MQ can read it when the key repository is accessed. The password must also be encrypted to reduce the likelihood of unauthorized access to the key repository.

"Working with SSL/TLS on IBM i" on page 270
This collection of topics gives instructions for individual tasks working with Transport Layer Security (TLS) in IBM MQ for IBM i.

IBM i *Creating a certificate authority and certificate for testing on IBM i*

Use this procedure to create a local CA certificate to sign certificate requests, and to create and install the CA certificate.

Before you begin

The instructions in this topic assume that a local certificate authority (CA) does not exist. If a local CA does exist, go to "Requesting a server certificate on IBM i" on page 280.

About this task

The CA certificates that are provided when you install TLS are signed by the issuing CA. On IBM i, you can generate a local certificate authority that can sign server certificates for testing TLS communications on your system. Follow these steps in a Web browser to create a local CA certificate:

Procedure

1. Access the DCM interface, as described in "Accessing DCM" on page 270.
2. In the navigation panel, click **Create a Certificate Authority**.

   The Create a Certificate Authority page is displayed in the task frame.
3. Type a password in the **Certificate store password** field and type it again in the **Confirm password** field.
4. Type a name in the **Certificate Authority (CA) name** field, for example `TLS Test Certificate Authority`.
5. Type appropriate values in the **Common Name** and **Organization** fields, and select a country. For the remaining optional fields, type the values you require.
6. Type a validity period for the local CA in the **Validity period** field.

   The default value is 1095 days.
7. Click **Continue**.

The CA is created, and DCM creates a certificate store and a CA certificate for your local CA.

8. Click **Install certificate**.

   The download manager dialog box is displayed.
9. Type the full path name for the temporary file in which you want to store the CA certificate and click **Save**.
10. When download is complete, click **Open**.

    The Certificate window is displayed.
11. Click **Install certificate**.

    The Certificate Import wizard is displayed.
12. Click **Next**.
13. Select **Automatically select the certificate store based on the type of certificate** and click **Next**.
14. Click **Finish**.

    A confirmation window is displayed.
15. Click **OK**.
16. In the Certificate window, click **OK**.
17. Click **Continue**.

    The Certificate Authority Policy page is displayed in the task frame.
18. In the **Allow creation of user certificates** field, select **Yes**.
19. In the **Validity period** field, type the validity period of certificates that are issued by your local CA.

    The default value is 365 days.
20. Click **Continue**.

    The Create a Certificate in New Certificate Store page is displayed in the task frame.
21. Check that none of the applications are selected.
22. Click **Continue** to complete the setup of the local CA.

What to do next

If you need to renew an existing certificate, see Renewing an existing certificate in the IBM i documentation.

IBM i *Requesting a server certificate on IBM i*

Digital certificates protect against impersonation, certifying that a public key belongs to a specified entity. A new server certificate can be requested from a certificate authority using the Digital Certificate Manager (DCM).

About this task

Perform the following steps in a Web browser:

Procedure

1. Access the DCM interface, as described in "Accessing DCM" on page 270.
2. In the navigation panel, click **Select a Certificate Store**.

   The Select a Certificate Store page is displayed in the task frame.
3. Select the certificate store you want to use and click **Continue**.
4. Optional: If you selected ***SYSTEM** in step 3, enter the system store password and click **Continue**.
5. Optional: If you selected **Other System Certificate Store** in step 3, in the **Certificate store path and filename** field, type the IFS path and file name you set when you created your certificate store. Also type a password in the **Certificate Store Password** field. Then click **Continue**
6. In the navigation panel, click **Create Certificate**.
7. In the task frame, select the **Server or client certificate** radio button and click **Continue**.

The Select a Certificate Authority (CA) page is displayed in the task frame.

8. If you have a local CA on your workstation you choose either the local CA or a commercial CA to sign the certificate. Select the radio button for the CA you want and click **Continue**.

   The Create a Certificate page is displayed in the task frame.
9. Optional: For a queue manager, in the **Certificate label** field, enter the certificate label.

   The label is either the value of the **CERTLABL** attribute, if it is set, or the default `ibmwebspheremq` with the name of the queue manager appended, all in lowercase. See Digital certificate labels for details.

   For example, for queue manager QM1, type `ibmwebspheremqqm1` to use the default value.
10. Optional: For an IBM MQ MQI client, in the **Certificate label** field, type `ibmwebspheremq` followed by your logon user ID folded to lowercase.
    For example, type `ibmwebspheremqmyuserID`
11. Type appropriate values in the **Common Name** and **Organization** fields, and select a country. For the remaining optional fields, type the values you require.

Results

If you selected a commercial CA to sign your certificate, DCM creates a certificate request in PEM (Privacy-Enhanced Mail) format. Forward the request to your chosen CA.

If you selected the local CA to sign your certificate, DCM informs you that the certificate has been created in the certificate store and can be used.

IBM i *Requesting a server certificate for a remote system on IBM i*

Follow this procedure to create a certificate signed by your local certificate authority (CA), or to apply for a server certificate signed by a commercial CA for import into a key repository on other platforms.

About this task

A user certificate must be used when the Digital Certificate Manager (DCM) serves as the certificate manager for IBM MQ on multiple platforms. For personal certificates that are distributed to other platforms and imported into a key repository, perform the following steps in a web browser:

Procedure

1. Access the DCM interface, as described in "Accessing DCM" on page 270.
2. In the **navigation** pane, click **Create Certificate**.

   The **Create Certificate** page is displayed in the task frame.
3. On the **Create Certificate** panel, select the **User certificate** radio button and click **Continue**.

   The **Create User Certificate** page is displayed.
4. On the **Create User Certificate** panel, complete the required fields under Certificate Information for **Organization name**, **State** or **province**, **Country** or **region**. Optionally, put values in the **Organization unit** and **Locality** or **city** fields. Click **Continue**.

   The **Common name** is automatically set to the user ID with which you are logged on to the iSeries system.
5. On the next **Create User Certificate** panel, click **Install certificate** and click **Continue**.

   A message is displayed stating, `Your personal certificate has been installed. You should keep a backup copy of this certificate.`
6. Click **OK**.
7. Depending on the web browser that you used to access DCM, complete one of the following steps:
   - For Microsoft Edge choose: **Tools>Internet Options>Content tab>Certificates button>Personal tab>**. Select the certificate and click **Export**.

- For Mozilla Firefox choose: **Tools>Options>Advanced>Encryption tab>View Certificates button>Your Certificates tab>**. Select the certificate and click **Backup**. Select the path and filename and click **OK**.

8. Transfer the exported certificate to the remote system using FTP in binary format.
9. Import the certificate that was exported in step "7" on page 281 to the key repository on the remote system.

   - If the certificate was saved using Microsoft Edge, use the instructions described in "Importing a personal certificate from a Microsoft .pfx file" on page 541 file.
   - If the certificate was saved using Mozilla Firefox, use the instructions described in Importing a personal certificate into a key repository.

   During the import, ensure that the label name of the personal certificate and the signer certificate are changed to the value that IBM MQ expects. The label must be either the value of the IBM MQ queue manager **CERTLABL** attribute, if it is set, or the default value of `ibmwebspheremq` with the name of the queue manager appended, all in lowercase. For more information, see Digital certificate labels.

IBM i *Adding server certificates to a key repository on IBM i*

Follow this procedure to add a requested certificate to the key repository.

About this task

After the CA sends you a new server certificate, you add it to the certificate store from which you generated the request. If the CA sends the certificate as part of an email message, copy the certificate into a separate file.

Note:

- You do not need to perform this procedure if the server certificate is signed by your local CA.
- Before you import a server certificate in PKCS #12 format into DCM, you must first import the corresponding CA certificate.

Use the following procedure to receive a server certificate into the queue manager certificate store:

Procedure

1. Access the DCM interface, as described in "Accessing DCM" on page 270.
2. In the **Manage Certificates** task category in the navigation panel, click **Import Certificate**.

   The Import Certificate page displays in the task frame.
3. Select the radio button for your certificate type and click **Continue**.

   Either the Import Server or Client Certificate page or the Import Certificate Authority (CA) Certificate page displays in the task frame.
4. In the **Import File** field, type the file name of the certificate you want to import and click **Continue**.

   DCM automatically determines the format of the file.
5. If the certificate is a **Server or client** certificate, type the password in the task frame and click **Continue**.

   DCM informs you that the certificate has been imported.

IBM i *Exporting a certificate from a key repository on IBM i*

Exporting a certificate exports both the public and private key. This action should be taken with extreme caution, since passing on a private key would completely compromise your security.

Before you begin

When you share a user's certificate with another user, you exchange public keys. This process is described in **Task 5. Sharing Certificates** in the Sharing Certificates section of "Quick Start Guide for AMS on AIX and Linux" on page 598. When you export a certificate as described here, you export both the public and private key. This action should be taken with extreme caution, since passing on a private key would completely compromise your security.

About this task

Perform the following steps on the computer from which you want to export the certificate:

Procedure

1. Access the DCM interface, as described in "Accessing DCM" on page 270.
2. In the navigation panel, click **Select a Certificate Store**.

   The Select a Certificate Store page is displayed in the task frame.
3. Select the certificate store you want to use and click **Continue**.
4. Optional: If you selected ***SYSTEM** in step 3, enter the system store password and click **Continue**.
5. Optional: If you selected **Other System Certificate Store** in step 3, in the **Certificate store path and filename** field, type the IFS path and file name you set when you created your certificate store and type a password in the **Certificate Store Password** field. Then click **Continue**
6. In the **Manage Certificates** task category in the navigation panel, click **Export Certificate**.

   The Export a Certificate page is displayed in the task frame.
7. Select the radio button for your certificate type and click **Continue**.

   Either the Export Server or Client Certificate page or the Export Certificate Authority (CA) Certificate page is displayed in the task frame.
8. Select the certificate you want to export.
9. Select the radio button to specify whether you want to export the certificate to a file or directly into another certificate store.
10. If you selected to export a server or client certificate to a file, provide the following information:
    - The path and file name of the location where you want to store the exported certificate.
    - For a personal certificate, the password that is used to encrypt the exported certificate and the target release. For CA certificates, you do not need to specify the password.
11. If you selected to export a certificate directly into another certificate store, specify the target certificate store and its password.
12. Click **Continue**.

IBM i *Importing a certificate into a key repository on IBM i*

Follow this procedure to import a certificate.

Before you begin

Before you import a personal certificate in PKCS #12 format into DCM, you must first import the corresponding CA certificate.

About this task

Perform these steps on the machine to which you want to import the certificate.

Procedure

1. Access the DCM interface, as described in "Accessing DCM" on page 270.
2. In the navigation panel, click **Select a Certificate Store**.

   The Select a Certificate Store page is displayed in the task frame.
3. Select the certificate store you want to use and click **Continue**.
4. Optional: If you selected ***SYSTEM** in step 3, enter the system store password and click **Continue**.

5. Optional: If you selected **Other System Certificate Store** in step 3, in the **Certificate store path and filename** field, type the IFS path and file name you set when you created your certificate store and type a password in the **Certificate Store Password** field. Then click **Continue**
6. In the **Manage Certificates** task category in the navigation panel, click **Import Certificate**.

   The Import Certificate page is displayed in the task frame.
7. Select the radio button for your certificate type and click **Continue**.

   Either the Import Server or Client Certificate page or the Import Certificate Authority (CA) Certificate page is displayed in the task frame.
8. In the **Import File** field, type the file name of the certificate you want to import and click **Continue**.

   DCM automatically determines the format of the file.
9. If the certificate is a **Server or client** certificate, type the password in the task frame and click **Continue**. DCM informs you that the certificate has been imported.

IBM i *Removing certificates in IBM i*

Use this procedure to remove personal certificates.

Procedure

1. Access the DCM interface, as described in "Accessing DCM" on page 270.
2. In the navigation panel, click **Select a Certificate Store**.

   The Select a Certificate Store page is displayed in the task frame.
3. Select the **Other System Certificate Store** check box and click **Continue**.

   The Certificate Store and Password page is displayed.
4. In the **Certificate store path and filename** field, type the IFS path and file name you set when you created the certificate store.
5. Type a password in the **Certificate Store Password** field. Click **Continue**.

   The Current Certificate Store page is displayed in the task frame.
6. In the **Manage Certificates** task category in the navigation panel, click **Delete Certificate**.

   The Confirm Delete Certificate page is displayed in the task frame.
7. Select the certificate you want to delete. Click **Delete**.
8. Click **Yes** to confirm that you want to delete the certificate. Otherwise, click **No**.

   DCM informs you if it has deleted the certificate.

IBM i *Using the *SYSTEM certificate store for one-way authentication on IBM i*

Follow these instructions to set up one-way authentication.

Before you begin

- Create a queue manager, channels, and transmission queues.
- Create a server or client certificate on the server queue manager.
- Transfer the CA certificate to the client queue manager and imported it into the key repository.
- Start a listener on the server and client queue managers.

About this task

To use one-way authentication, using a computer running IBM i as the TLS server, set the SSL Key Repository (SSLKEYR) parameter to *SYSTEM. This setting registers the IBM MQ queue manager as an application. You can then assign a certificate to the queue manager to enable one-way authentication.

You can also use private keystores to implement one-way authentication by creating a dummy certificate for the client queue manager in the key repository.

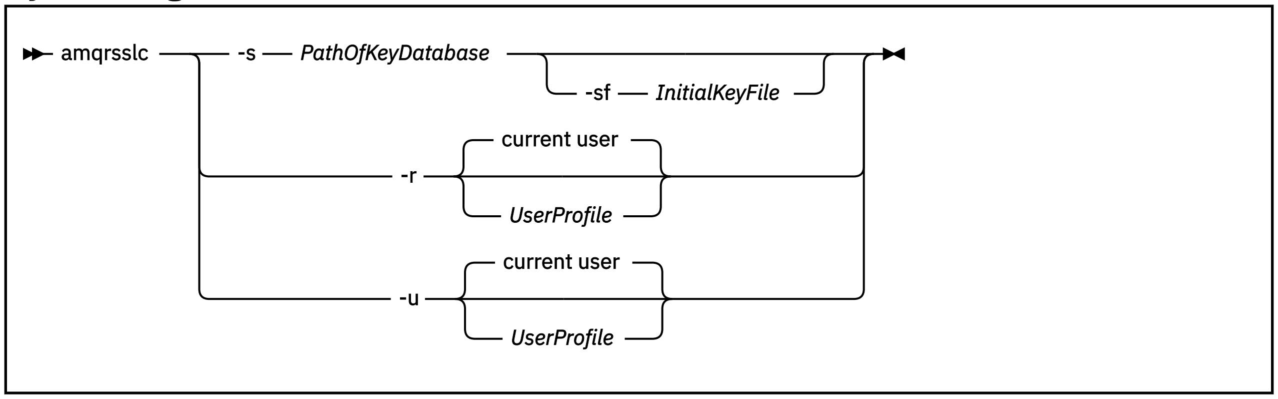
Procedure

1. Perform the following steps on the server and client queue managers:
   a) Alter the queue manager to set the SSLKEYR parameter by issuing the command `CHGMQM MQMNAME(SSL) SSLKEYR(*SYSTEM)`.
   b) Stash the password for the default key repository by issuing the command `CHGMQM MQMNAME(SSL) SSLKEYRPWD('xxxxxxx')`.
      The password must be in single quotation marks.
   c) Alter the channels to have the correct CipherSpec in the SSLCIPHER parameter.
   d) Refresh TLS security by issuing the command `RFRMQMAUT QMNAME(QMGRNAME) TYPE(*SSL)`.
2. Assign the certificate to the server queue manager using DCM, as follows:
   a) Access the DCM interface, as described in "Accessing DCM" on page 270.
   b) In the navigation panel, click **Select a Certificate Store**.
      The Select a Certificate Store page is displayed in the task frame.
   c) Select the *SYSTEM certificate store and click **Continue**.
   d) In the left panel, expand **Manage Applications**.
   e) Select the **View Application** definition to check that the queue manager has been registered as an application.
      `SSL (WMQ)` is listed in the table.
   f) Select **Update Certificate Assignment**.
   g) Select **Server** and click **Continue**.
   h) Select QMGRNAME (WMQ) and click **Update certificate assignment**.
   i) Select the certificate and click **Assign New Certificate**. A window opens stating that the certificate has been assigned to the application.

IBM i *IBM MQ SSL Client utility (amqrsslc) for IBM i*

The IBM MQ SSL Client utility (amqrsslc) for IBM i is used by the IBM MQ MQI client on IBM i systems to register or unregister the client user profile, or stash the certificate store password. The utility can only be run by a user with a profile with *ALLOBJ special authority or a member of QMQMADM that has options to create or delete application registrations in the Digital Certificate Manager (DCM).

Syntax diagram



Register the client user profile

If the IBM MQ MQI client is using the *SYSTEM certificate store, you must register the client user profile (logon user) for use as an application with Digital Certificate Manager (DCM).

If you want to register the client user profile, run the **`amqrsslc`** program with the `-r` option with *UserProfile*. The user profile used when calling **`amqrsslc`** must have *USE authority. Providing *UserProfile* with the `-r` option registers the *UserProfile* as a server application with a unique application label of QIBM_WEBSPHERE_MQ_*UserProfile* and a label with a description of *UserProfile* (WMQ). This server application then is displayed in the DCM, and you can assign to this application any server or client certificate in the system store.

Note: If a user profile is not specified with `-r` option, then the user profile of the user running the **`amqrsslc`** tool is registered.

The following code uses **`amqrsslc`** to register a user profile. In the first example, the specified user profile is registered; in the second it is the profile of the logged in user:

```
CALL PGM(QMQM/AMQRSSLC) PARM('-r' UserProfile)
CALL PGM(QMQM/AMQRSSLC) PARM('-r')
```

Unregister the client user profile

To unregister the client profile, run the **`amqrsslc`** program with the `-u` option with *UserProfile*. The user profile used when calling **`amqrsslc`** must have *USE authority. Providing the *UserProfile* with the `-u` option unregisters *UserProfile* with label QIBM_WEBSPHERE_MQ_*UserProfile* from the DCM.

Note: If a user profile is not specified with `-u` option, then the user profile of the user running the **`amqrsslc`** tool is unregistered.

The following code uses **`amqrsslc`** to unregister a user profile. In the first example, the specified user profile is unregistered; in the second it is the profile of the logged in user:

```
CALL PGM(QMQM/AMQRSSLC) PARM('-u' UserProfile)
CALL PGM(QMQM/AMQRSSLC) PARM('-u')
```

Stash the certificate store password

If the IBM MQ MQI client is not using the *SYSTEM certificate store and using another certificate store (that is, MQSSLKEYR is set to value other than *SYSTEM), then the password of the key database can be stashed so that it does not need to be specified by the client application when it runs.

Use the `-s` option to stash the password of the key database. Specify the full path and name of the key database. If the file extension is not supplied, it is assumed to be `.kdb`.

In the following code, the fully qualified file name of the certificate store is `/Path/Of/KeyDatabase/MyKey.kdb`:

```
CALL PGM(QMQM/AMQRSSLC) PARM('-s' '/Path/Of/KeyDatabase/MyKey')
```

Running this code results in a request for the password of this key database. This password is stashed in a file with the same name as key database with an `.sth` extension.

Additionally, the initial key to encrypt the password can be specified. The initial key should be stored in a file as a single line of text and then the location of that file is supplied to the program through the **`-sf`** flag. If no initial key file is supplied a default key is used to encrypt the password.

The stash file is stored on the same path as the key database. The code example generates a stash file of `/Path/Of/KeyDatabase/MyKey.sth`.

QMQM is the user owner and QMQMADM the group owner for this file. QMQM and QMQMADM have read, write permission, and other profiles have only read permission.

IBM i *When changes to certificates or the certificate store become effective on IBM i*

When you change the certificates in a certificate store, or the location of the certificate store, the changes take effect depending on the type of channel and how the channel is running.

Changes to the certificates in the certificate store and to the key repository attribute become effective in the following situations:

- When a new outbound single channel process first runs a TLS channel.
- When a new inbound TCP/IP single channel process first receives a request to start a TLS channel.
- When the MQSC command REFRESH SECURITY TYPE(SSL) is issued to refresh the IBM MQ TLS environment.
- For client application processes, when the last TLS connection in the process is closed. The next TLS connection picks up the certificate changes.
- For channels that run as threads of a process pooling process (amqrmppa), when the process pooling process is started or restarted and first runs a TLS channel. If the process pooling process has already run a TLS channel, and you want the change to become effective immediately, run the MQSC command REFRESH SECURITY TYPE(SSL).
- For channels that run as threads of the channel initiator, when the channel initiator is started or restarted and first runs a TLS channel. If the channel initiator process has already run a TLS channel, and you want the change to become effective immediately, run the MQSC command REFRESH SECURITY TYPE(SSL).
- For channels that run as threads of a TCP/IP listener, when the listener is started or restarted and first receives a request to start a TLS channel. If the listener has already run a TLS channel, and you want the change to become effective immediately, run the MQSC command REFRESH SECURITY TYPE(SSL).

IBM i *Configuring cryptographic hardware on IBM i*

Use this procedure to configure the Cryptographic Coprocessor on IBM i

Before you begin

Ensure your user profile has *ALLOBJ and *SECADM special authorities to enable you to configure the coprocessor hardware.

Procedure

1. Go to either `http://`*`machine`*`.domain:2001` or `https://`*`machine`*`.domain:2010`, where *machine* is the name of your computer.

   A dialog box is displayed, requesting a user name and a password.
2. Type a valid IBM i user profile and password.
3. Go to Cryptography and follow the appropriate links for further information.

What to do next

For more specific information about configuring the 4767 Cryptographic Coprocessor, see 4767 Cryptographic Coprocessor.

ALW Working with SSL/TLS on AIX, Linux, and Windows

On AIX, Linux, and Windows systems, Transport Layer Security (TLS) support is installed with IBM MQ.

Note: Deprecated V 9.4.0 From IBM MQ 9.4.0, the use of CMS key repositories and stash files with IBM MQ Java applications is deprecated. Migrate to using PKCS #12 key repositories and protect the key repository passwords by using the IBM MQ password protection system.

Important: V 9.4.0 V 9.4.0 From IBM MQ 9.4.0, CMS key repositories and stash files are not supported with AMQP and MQTT channels that use SSL/TLS. Use PKCS #12 key repositories and protect the key repository passwords by using the IBM MQ password protection system instead.

For more detailed information about certificate validation policies, see Certificate validation and trust policy design.

For more information about the commands that are used to manage key repositories and certificates on AIX, Linux, and Windows, see "runmqakm and runmqktool commands on AIX, Linux, and Windows" on page 528.

ALW *Setting up a key repository on AIX, Linux, and Windows*

Follow this procedure to create a new key repository.

Before you begin

A key repository is secured with a password as it contains sensitive information. Before you create the key repository, review the options that IBM MQ provides to securely store the key repository password. For more information, see "Encrypting key repository passwords on AIX, Linux, and Windows" on page 290.

Note: Deprecated V 9.4.0 From IBM MQ 9.4.0, the use of CMS key repositories and stash files with IBM MQ Java applications is deprecated. Migrate to using PKCS #12 key repositories and protect the key repository passwords by using the IBM MQ password protection system.

Important: V 9.4.0 V 9.4.0 From IBM MQ 9.4.0, CMS key repositories and stash files are not supported with AMQP and MQTT channels that use SSL/TLS. Use PKCS #12 key repositories and protect the key repository passwords by using the IBM MQ password protection system instead. You can create a PKCS #12 key repository by using the following command:

```
runmqakm -keydb -create -db filename.p12 -pw password -type pkcs12
```

This command creates a PKCS #12 key repository file named *filename*.p12 that is secured with the specified password.

About this task

A TLS connection requires a *key repository* at each end of the connection. Each IBM MQ queue manager and IBM MQ MQI client must have access to a key repository. For more information, see "The SSL/TLS key repository" on page 25.

Digital certificates are stored in the key repository. These digital certificates have labels. The certificate label associates a personal certificate with a specific queue manager or IBM MQ MQI client. TLS uses that certificate for authentication purposes. On AIX, Linux, and Windows systems, IBM MQ uses one of the following values for the certificate label:

- The value of the **CERTLABL** queue manager or channel attribute, if it is set.
- The default value of `ibmwebspheremq`, with the name of the queue manager or IBM MQ MQI client user logon ID appended, all in lowercase.

For more information, see Digital certificate labels.

The key repository file name comprises a path and stem name:

- On AIX and Linux systems, the default path for a queue manager (set when you created the queue manager) is `/var/mqm/qmgrs/`*`queue_manager_name`*`/ssl`.

  On Windows systems, the default path is *`MQ_DATA_PATH`*`\qmgrs\`*`queue_manager_name`*`\ssl`, where *`MQ_DATA_PATH`* is the data path that is selected during the installation of IBM MQ. For example, `C:\ProgramData\IBM\MQ\qmgrs\QM1\ssl`.

  The default file name is `key.kdb`. Alternatively, you can use your own path and file name.

  If you choose your own path or file name, set the permissions to the file to tightly control access to it.

- For an IBM MQ client, there is no default path or file name. Tightly control access to this file.

Do not create key repositories on a file system that does not support file level locks, for example NFS version 2 on Linux systems.

See "Changing the key repository location for a queue manager on AIX, Linux, and Windows" on page 294 for information about checking and specifying the key database file name. You can specify the key database file name either before or after the key repository is created.

You can use the **runmqakm** (GSKCapiCmd) or the V 9.4.0 V 9.4.0 **runmqktool** (keytool) commands to manage key repositories that are used by IBM MQ. For more information, see "runmqakm and runmqktool commands on AIX, Linux, and Windows" on page 528.

The user ID that runs the commands to manage the key repository must have write permission for the directory in which the key repository file is created or updated. For a queue manager that uses the default `ssl` directory, the user ID that runs the **runmqakm** or **runmqktool** command must be a member of the mqm group. For an IBM MQ MQI client, if you run **runmqakm** or **runmqktool** from a user ID that is different to the user ID which runs the client, you must alter the file permissions to enable the IBM MQ MQI client to access the key repository. For more information, see "Accessing and securing your key database files on Windows" on page 292 or "Accessing and securing your key database files on AIX and Linux systems" on page 292.

You can create a new, empty, key repository by using the **runmqakm** command.
V 9.4.0 V 9.4.0 If you use the **runmqktool** command instead, the key repository is created when a command is issued to create or import a certificate.

Note: If you must manage TLS certificates in a way that is FIPS-compliant, use the **runmqakm** command.

Procedure

1. Issue the following command to create a key repository with the **runmqakm** command:

```
runmqakm -keydb -create -db filename -pw password -type type
         -stash -fips -strong
```

where:

-db *filename*
Specifies the fully qualified file name of the key repository.

-pw *password*
Specifies the password for the key repository.

-type *type*
V 9.4.0 V 9.4.0 Specifies the type of key repository. For a key repository that is used by IBM MQ, the possible values are:

- `pkcs12`
- Deprecated `cms`

Note: From IBM MQ 9.4.0, the use of CMS key repositories and stash files is deprecated for IBM MQ Java applications and is not supported for AMQP and MQTT channels that use SSL/TLS.

-stash
Optional. Specify this option to store the key repository password in a stash file. You do not need to store the password in a stash file if you encrypt the password using the IBM MQ password protection system instead.

-fips
Specifies that the command is run in FIPS mode. When in FIPS mode, the IBM Crypto for C (ICC) component uses algorithms that are FIPS 140-2 validated. If the ICC component does not initialize in FIPS mode, the **runmqakm** command fails.

-strong
Checks that the password entered satisfies the minimum requirements for password strength. The minimum requirements for a password are as follows:

- The password must be a minimum length of 14 characters.
- The password must contain a minimum of one lowercase character, one uppercase character, and one digit or special character. Special characters include the asterisk (*), the dollar sign ($), the number sign (#), and the percent sign (%). A space is classified as a special character.
- Each character can occur a maximum of three times in a password.
- A maximum of two consecutive characters in the password can be identical.
- All characters are in the standard ASCII printable character set, within the range 0x20 - 0x7E.

2. Set the access permissions for the key repository files as described in "Accessing and securing your key database files on Windows" on page 292 or "Accessing and securing your key database files on AIX and Linux systems" on page 292.

   On Windows, by default only the user ID that ran the command to create the key repository is granted access to read the stash (`.sth`) file. After a stash file is created with the **runmqakm** command, check the file permissions, and grant permission to the service account running the queue manager, or to a group such as local `mqm`.
3. If you are not using a stash file, provide the keystore password to the queue manager or client application by following the instructions in "Supplying the key repository password for a queue manager on AIX, Linux, and Windows" on page 294 or "Supplying the key repository password for an IBM MQ MQI client on AIX, Linux, and Windows" on page 296.

What to do next
Add default certificate authority (CA) certificates to the empty key repository, if required. For more information, see "Adding default CA certificates into an empty key repository on AIX, Linux, and Windows" on page 293.

ALW *Generating strong passwords for key repository protection on AIX, Linux, and Windows*
You can generate strong passwords for key repository protection using the **runmqakm** (GSKCapiCmd) command.

You can use the **runmqakm** command with the following parameters to generate a strong password:

```
runmqakm -random -create -length password_length -strong -fips
```

where *password_length* is the length of the password to generate. The minimum password length that can be specified is 14.

When using the generated password on the **-pw** parameter of subsequent certificate administration commands, always place double quotation marks around the password. On AIX and Linux systems, you must also use a backslash character to escape the following characters if they appear in the password string:

```
!  \  "  '
```

When you enter a key repository password in response to a prompt from the **runmqakm** or V 9.4.0 V 9.4.0 **runmqktool** command, it is not necessary to quote or escape the password as the operating system shell does not affect data entry in these cases.

ALW *Encrypting key repository passwords on AIX, Linux, and Windows*
Several IBM MQ components need access to a key repository that contains digital certificates or symmetric keys. A key repository is secured with a password as it contains sensitive information. The key repository password must be stored in a location where IBM MQ can read it when the key repository

is accessed. The password must also be encrypted to reduce the likelihood of unauthorized access to the key repository.

The following IBM MQ components and features support two different methods to store key repository passwords:

- The queue manager TLS key repository.
- IBM MQ MQI clients that use TLS.
- V 9.4.0 The Native HA configuration in the **NativeHALocalInstance** stanza of the `qm.ini` file.
- V 9.4.0 The token authentication configuration in the **AuthToken** stanza of the `qm.ini` file.

Key repository passwords for use by these components can be encrypted and stored by using one of the following methods:

The IBM MQ password protection system.

Each IBM MQ component provides a command to encrypt the key repository password. The encrypted command that the command outputs is stored in a file.

For the queue manager TLS key repository, the password is encrypted when the **SSLKEYRPWD** queue manager attribute is set.

The password is encrypted with the AES-128 algorithm. The details of this algorithm are publicly known and it is considered secure.

The password is stored in a proprietary format that is not understood by other software that might access the key repository.

A password that is encrypted by one IBM MQ component cannot be used by a different IBM MQ component.

A unique encryption key can be provided when the key repository password is encrypted. A unique encryption key prevents anyone who does not have access to the encryption key from being able to decrypt the password.

The plain text key repository password is needed to manage the certificates that are in the key repository. In addition to encrypting the key repository password by using the IBM MQ password protection system, you must also store the key repository password in a secure location where it can be accessed for this purpose.

For more information about the IBM MQ password protection system, see "Protecting passwords in IBM MQ component configuration files" on page 551.

A key repository stash file.

The **runmqakm** command can store the key repository password in a stash file.

The password is encrypted with a proprietary method that is specific to IBM MQ's cryptographic provider, IBM Global Security Kit (GSKit).

A unique encryption key cannot be provided.

The encrypted password is stored in a stash file in the same directory as the key repository file.

Anyone with read access to both the key repository and the stash file can access and manage the contents of the key repository.

Note: Deprecated V 9.4.0 From IBM MQ 9.4.0, the use of stash files with IBM MQ Java applications is deprecated.

Important: V 9.4.0 V 9.4.0 From IBM MQ 9.4.0, stash files are not supported by AMQP and MQTT channels that use TLS.

Regardless of the method that you choose to encrypt the key repository password, ensure that you are aware of the limitations of encrypting stored passwords. For more information, see "The limits to protection through password encryption" on page 558.

Related concepts

"Supplying the key repository password for a queue manager on AIX, Linux, and Windows" on page 294
As the key repository contains sensitive information, it is secured with a password. To be able to access the key repository contents to perform TLS operations, IBM MQ must be able to retrieve the key repository password.

"Supplying the key repository password for an IBM MQ MQI client on AIX, Linux, and Windows" on page 296
As the key repository contains sensitive information, it is secured with a password. To be able to access the key repository contents to perform TLS operations, IBM MQ must be able to retrieve the key repository password.

"Working with SSL/TLS on AIX, Linux, and Windows" on page 287
On AIX, Linux, and Windows systems, Transport Layer Security (TLS) support is installed with IBM MQ.

Windows *Accessing and securing your key database files on Windows*
The key database files might not have appropriate access permissions. You must set appropriate access to these files.

Set access control to the files *key*.p12, *key*.kdb, *key*.sth, *key*.crl, and *key*.rdb, where *key* is the stem name of your key database, to grant authority to a restricted set of users.

If you have used a different key repository extension other than .p12 or .kdb, you must ensure also that the permissions of this file are set.

Consider granting access as follows:

full authority
BUILTIN\Administrators, NT AUTHORITY\SYSTEM, and the user who created the database files.

read authority
For a queue manager, the local mqm group only. This assumes that the MCA is running under a user ID in the mqm group.

For a client, the user ID under which the client process is running.

Linux AIX *Accessing and securing your key database files on AIX and Linux systems*
The key database files might not have appropriate access permissions. You must set appropriate access to these files.

For a queue manager, set permissions on the key database files so that queue manager and channel processes can read them when necessary, but other users cannot read or modify them. Normally, the mqm user needs read permissions. If you have created the key database file by logging in as the mqm user, then the permissions are probably sufficient; if you were not the mqm user, but another user in the mqm group, you probably need to grant read permissions to other users in the mqm group.

Similarly for a client, set permissions on the key database files so that client application processes can read them when necessary, but other users cannot read or modify them. Normally, the user under which the client process runs needs read permissions. If you have created the key database file by logging in as that user, then the permissions are probably sufficient; if you were not the client process user, but another user in that group, you probably need to grant read permissions to other users in the group.

Set the permissions on the files *key*.p12, *key*.kdb, *key*.sth, *key*.crl, and *key*.rdb, where *key* is the stem name of your key database, to read and write for the file owner, and to read for the mqm or client user group (-rw-r-----).

If you have used a different key repository extension other than .p12 or .kdb, you must ensure also that the permissions of this file are set.

ALW *Adding default CA certificates into an empty key repository on AIX, Linux, and Windows*
Follow this procedure to add one or more of the default certificate authority (CA) certificates to an empty key repository.

When you create a new key repository, it is empty. You can add default CA certificates to a key repository by using the **runmqakm** command.

Using runmqakm

Issue the following command to add default CA certificates to a key repository with the **runmqakm** command:

```
runmqakm -cert -populate -db filename -pw password
```

where:

-db *filename*
Specifies the fully qualified file name of the key repository.

-pw *password*
Specifies the password for the key repository.

Note: IBM MQ trusts all certificates that are signed by the CA certificates in your key repository. Consider carefully which certificate authorities you want to trust and add only the CA certificates that are needed to authenticate your clients and queue managers. It is not recommended to add the full set of default CA certificates to a key repository.

ALW *Locating the key repository for a queue manager on AIX, Linux, and Windows*

Use this procedure to obtain the location of your queue manager's key database file

Procedure

1. Display your queue manager's attributes, using either of the following MQSC commands:

   ```
   DISPLAY QMGR ALL
   DISPLAY QMGR SSLKEYR
   ```

   You can also display your queue manager's attributes using the IBM MQ Explorer or PCF commands.
2. Examine the command output for the path and stem name of the key database file.
   For example,
   a. on AIX and Linux: `/var/mqm/qmgrs/QM1/ssl/key`, where `/var/mqm/qmgrs/QM1/ssl` is the path and `key` is the stem name
   b. on Windows: *MQ_INSTALLATION_PATH*`\qmgrs\QM1\ssl\key`, where *MQ_INSTALLATION_PATH*`\qmgrs\QM1\ssl` is the path and `key` is the stem name. *MQ_INSTALLATION_PATH* represents the high-level directory in which IBM MQ is installed.

   Note: From IBM MQ 9.3.0 the SSLKEYR field supports both a full filename (including extension) and a stem name (without extension). If a stem name is set, IBM MQ automatically appends `.kdb` and uses that key repository.

ALW Changing the key repository location for a queue manager on AIX, Linux, and Windows

You can change the location of your queue manager's key database file by various means including the MQSC command ALTER QMGR.

You can change the location of your queue manager's key database file by using the MQSC command ALTER QMGR to set your queue manager's key repository attribute. For example, on AIX and Linux:

```
ALTER QMGR SSLKEYR('/var/mqm/qmgrs/QM1/ssl/MyKey.kdb')
```

On Windows:

```
ALTER QMGR SSLKEYR('C:\Program Files\IBM\MQ\Qmgrs\QM1\ssl\Mykey.kdb')
```

Attention: On Windows and Linux, if TLS AMQP channels are used, the suffix of the key repository file must be one of the following:

- .kdb, for a CMS key repository
- .p12 or .pkcs12, for a PKCS #12 key repository.

You can also alter your queue manager's attributes using the IBM MQ Explorer or PCF commands.

When you change the location of a queue manager's key database file, certificates are not transferred from the old location. If the key database file you are now accessing is a new key database file, you must populate it with the CA and personal certificates you need, as described in "Importing a personal certificate into a key repository on AIX, Linux, and Windows" on page 539.

ALW Supplying the key repository password for a queue manager on AIX, Linux, and Windows

As the key repository contains sensitive information, it is secured with a password. To be able to access the key repository contents to perform TLS operations, IBM MQ must be able to retrieve the key repository password.

IBM MQ provides two mechanisms to supply the key repository password to a queue manager:

- "The KEYRPWD attribute" on page 294
- "The key repository stash file" on page 295

If you do not use a key repository stash file, the key repository password is encrypted by using the IBM MQ password protection system. For more information about the methods of protecting the key repository password, see "Encrypting key repository passwords on AIX, Linux, and Windows" on page 290.

The KEYRPWD attribute

To supply a key repository password directly to the queue manager, run the following MQSC command, replacing *password* with your key repository password:

```
ALTER QMGR KEYRPWD('password')
```

Attention: Ensure that you surround the password with single quotation marks, otherwise IBM MQ converts the characters to uppercase.

When a key repository password is specified by using this method, the password is encrypted by using the IBM MQ password protection system before it is stored.

An encryption key, which is known as the initial key, is used to encrypt the password. Set the queue manager to use a unique initial key to securely protect the password. If you do not supply an initial key, the default key is used.

Ensure that the queue manager is configured with a unique initial key before you set the key repository password. You can modify the initial key by using the **INITKEY** attribute on the **ALTER QMGR** command. For example:

```
ALTER QMGR INITKEY('mykey')
```

Warning: Modifying the initial key after setting the key repository password does not cause the key repository password to be encrypted with the new initial key. Changing the initial key without also resetting the key repository password results in IBM MQ being unable to decrypt the key repository password and, therefore, unable to access the key repository.

For more information about the **KEYRPWD** attribute, see KEYRPWD.

The key repository stash file

If a key repository password is not supplied to the queue manager by using the **KEYRPWD** attribute, IBM MQ assumes that a stash file exists in the same directory as the key repository. The stash file has the same stem name as the key repository, but has the .sth extension.

A key repository stash file is created at the same time as the key repository, or later, as a separate **runmqakm** command.

Attention: The format of the stash file is specific to the IBM MQ cryptographic provider IBM Global Security Kit (GSKit), and is not available on platforms that use a different cryptographic provider.

To create a stash file when the key repository is created, specify the **-stash** parameter. For example:

```
runmqakm -keydb -create -db key.kdb -pw passw0rd -stash
```

where *passw0rd* is the key repository password.

To create a stash file later, run the following command:

```
runmqakm -keydb -stashpw -db key.kdb -pw passw0rd
```

where *passw0rd* is the key repository password.

Related concepts

"Encrypting key repository passwords on AIX, Linux, and Windows" on page 290
Several IBM MQ components need access to a key repository that contains digital certificates or symmetric keys. A key repository is secured with a password as it contains sensitive information. The key repository password must be stored in a location where IBM MQ can read it when the key repository is accessed. The password must also be encrypted to reduce the likelihood of unauthorized access to the key repository.

"Supplying the key repository password for an IBM MQ MQI client on AIX, Linux, and Windows" on page 296
As the key repository contains sensitive information, it is secured with a password. To be able to access the key repository contents to perform TLS operations, IBM MQ must be able to retrieve the key repository password.

ALW *Locating the key repository for an IBM MQ MQI client on AIX, Linux, and Windows*

The location of the key repository is given by the MQSSLKEYR variable, or specified in the MQCONNX call.

Examine the MQSSLKEYR environment variable to find the location of the key database file for your IBM MQ MQI client. For example:

```
echo $MQSSLKEYR
```

Also check your application, because the key database file name can also be set in an MQCONNX call, as described in“Specifying the key repository location for an IBM MQ MQI client on AIX, Linux, and Windows” on page 296. The value set in an MQCONNX call overrides the value of MQSSLKEYR.

ALW *Specifying the key repository location for an IBM MQ MQI client on AIX, Linux, and Windows*

There is no default key repository for an IBM MQ MQI client. You can specify its location in either of two ways. Ensure that the key database file can be accessed only by intended users or administrators to prevent unauthorized copying to other systems.

You can specify the location of the key database file for your IBM MQ MQI client in two ways:

- Setting the MQSSLKEYR environment variable. For example, on AIX and Linux:

  ```
  export MQSSLKEYR=/var/mqm/ssl/key.kdb
  ```

  On Windows:

  ```
  set MQSSLKEYR=C:\Program Files\IBM\MQ\ssl\key.kdb
  ```

- Providing the path and stem name of the key database file in the *KeyRepository* field of the MQSCO structure when an application makes an MQCONNX call. For more information about using the MQSCO structure in MQCONNX, see Overview for MQSCO.

ALW *Supplying the key repository password for an IBM MQ MQI client on AIX, Linux, and Windows*

As the key repository contains sensitive information, it is secured with a password. To be able to access the key repository contents to perform TLS operations, IBM MQ must be able to retrieve the key repository password.

IBM MQ provides four mechanisms to supply the key repository password to a IBM MQ MQI client:

- “The KeyRepoPassword fields of MQSCO ” on page 296
- “The MQKEYRPWD environment variable” on page 297
- “The SSLKeyRepositoryPassword attribute of the client configuration file” on page 297
- “The key repository stash file” on page 297

If you do not use a key repository stash file, you can supply the key repository password as a plain text string, or a string that is encrypted by using the IBM MQ password protection system. For more information about the methods of protecting the key repository password, see “Encrypting key repository passwords on AIX, Linux, and Windows” on page 290.

The KeyRepoPassword fields of MQSCO

To supply a key repository password by using the MQSCO structure, you must use a combination of the following three variable string fields:

KeyRepoPasswordLength
: The length of the password.

KeyRepoPasswordPtr
: A pointer to the location in memory that contains the password.

KeyRepoPasswordOffset
: The location of the password in memory, represented as number of bytes from the start of the MQSCO structure.

Note: You can supply only one of **KeyRepoPasswordPtr** or **KeyRepoPasswordOffset**.

For example:

```
char * pwd = "passw0rd";
MQSCO  SslConnOptions = {MQSCO_DEFAULT};

SslConnOptions.KeyRepoPasswordPtr = pwd;
SslConnOptions.KeyRepoPasswordLength = (MQLONG)strlen(SslConnOptions.KeyRepoPasswordPtr);
SslConnOptions.Version = MQSCO_VERSION_6;
```

Attention: If you supply the password by using this method, encrypt the password before it is supplied to the IBM MQ client application. For more information, see "Encrypting the key repository password" on page 298.

For more information about the MQCSO structure, see MQSCO - SSL/TLS configuration options.

The MQKEYRPWD environment variable

If a key repository password is not supplied to the client by using the MQSCO structure, you can specify the key repository password by using the **MQKEYRPWD** environment variable. For example:

```
export MQKEYRPWD=passw0rd
```

or

```
set MQKEYRPWD=passw0rd
```

where `passw0rd` is your password.

Attention: If you supply the password by using this method, encrypt the password before you set the value of the environment variable. For more information, see "Encrypting the key repository password" on page 298.

The SSLKeyRepositoryPassword attribute of the client configuration file

If a key repository password is not supplied to the client by using one of the other methods, you can specify the key repository password by using the **SSLKeyRepositoryPassword** attribute in the **SSL** stanza of the client configuration file. For example:

```
SSL:
    SSLKeyRepositoryPassword=passw0rd
```

Attention: If you supply the password by using this method, encrypt the password before setting the value of the **SSLKeyRepositoryPassword** attribute. For more information, see "Encrypting the key repository password" on page 298.

Ford more information about the SSL stanza of the client configuration file, see SSL stanza of the client configuration file.

The key repository stash file

If the key repository password is not supplied to the client by using one of the other methods, IBM MQ assumes that a stash file exists in the same directory as the key repository. The stash file has the same stem name as the key repository, but has the `.sth` extension.

A key repository stash file is created at the same time as the key repository, or later, using a separate **runmqakm** command.

Attention: The format of the stash file is specific to the IBM MQ cryptographic provider IBM Global Security Kit (GSKit), and is not available on platforms that use a different cryptographic provider.

To create a stash file when the key repository is created, specify the **-stash** parameter. For example:

```
runmqakm -keydb -create -db key.kdb -pw passw0rd -stash
```

where *passw0rd* is the key repository password.

To create a stash file later, run the following command:

```
runmqakm -keydb -stashpw -db key.kdb -pw passw0rd
```

where *passw0rd* is the key repository password.

Encrypting the key repository password

If you supply the key repository password by using any method other than a stash file, encrypt the password by using the IBM MQ password protection system. To encrypt the password, run the **runmqicred** command. Enter the key repository password when prompted. The command outputs the encrypted password. The encrypted password can be supplied to the IBM MQ MQI client instead of the plain text password by using any of the methods described.

An encryption key, which is known as the initial key, is used to encrypt the password. When you encrypt the password, use a unique initial key to securely protect the password. To supply your own initial key, use the **-sf** parameter to the **runmqicred** command. If you do not supply an initial key, the default key is used.

For more information, see runmqicred (protect IBM MQ client passwords).

If you supply your own initial key when the key repository password is encrypted, and provide the encrypted password to the IBM MQ MQI client, you must also ensure that you supply the same initial key to the IBM MQ MQI client. For more information about how to provide the initial key to an IBM MQ MQI client, see "Supplying an initial key for an IBM MQ MQI client on AIX, Linux, and Windows" on page 298.

Related concepts

"Encrypting key repository passwords on AIX, Linux, and Windows" on page 290
Several IBM MQ components need access to a key repository that contains digital certificates or symmetric keys. A key repository is secured with a password as it contains sensitive information. The key repository password must be stored in a location where IBM MQ can read it when the key repository is accessed. The password must also be encrypted to reduce the likelihood of unauthorized access to the key repository.

"Supplying the key repository password for a queue manager on AIX, Linux, and Windows" on page 294
As the key repository contains sensitive information, it is secured with a password. To be able to access the key repository contents to perform TLS operations, IBM MQ must be able to retrieve the key repository password.

ALW *Supplying an initial key for an IBM MQ MQI client on AIX, Linux, and Windows*
If you supply variables to an IBM MQ MQI client that have been encrypted using the IBM MQ Password Protection System, you might need to supply the corresponding initial key that was used to encrypt the value.

If you did not specify an initial key when encrypting the value, you do not need to provide any initial key value to the IBM MQ client. However, if you used a unique initial key you can provide the initial key to the IBM MQ client using the following methods:

- "Supplying the initial key using the MQCSP structure" on page 298
- "Supplying the initial key using the MQS_MQI_KEYFILE environment variable" on page 299
- "Supplying the initial key using the client configuration file" on page 299

Supplying the initial key using the MQCSP structure

To supply the initial key using the MQCSP structure, you must use a combination of the following three variable string fields:

`InitialKeyLength`
The length of the initial key

`InitialKeyPtr`
A pointer to the location in memory containing the initial key

`InitialKeyOffset`
The location of the initial key in memory, represented as number of bytes from the start of the MQCSP structure.

Note: You can supply only one of **`InitialKeyPtr`** or **`InitialKeyOffset`**.

For example:

```
char * initialKey = "myInitialKey";
MQCSP  cspOptions = {MQCSP_DEFAULT};


cspOptions.InitialKeyPtr = initialKey;
cspOptions.InitialKeyLength = (MQLONG)strlen(cspOptions.InitialKeyPtr);
cspOptions.Version = MQCSP_VERSION_2;
```

Supplying the initial key using the MQS_MQI_KEYFILE environment variable

If an initial key is not supplied to the client using the MQCSP structure, IBM MQ checks the *MQS_MQI_KEYFILE* environment variable. You should set this environment variable to the location of a file containing a single line of text, consisting of the initial key you want to use.

For example, if a file called `mykey.key` exists in the root directory, and contains the initial key, you should set the environment variable as follows:

```
export MQS_MQI_KEYFILE=/mykey.key
```

or

```
set MQS_MQI_KEYFILE=C:\mykey.key
```

Supplying the initial key using the client configuration file

If an initial key is not supplied to the client using a previous mechanism, IBM MQ checks the **`MQIInitialKeyFile`** attribute of the Security stanza of the `mqclient.ini` file. You should set this attribute to the location of a file containing a single line of text, consisting of the initial key you want to use.

For example, if a file called `mykey.key` exists in the root directory, and contains the initial key, the client configuration file should contain the following:

```
Security:
    MQIInitialKeyFile=/mykey.key
```

Related concepts

"Supplying the key repository password for an IBM MQ MQI client on AIX, Linux, and Windows" on page 296
As the key repository contains sensitive information, it is secured with a password. To be able to access the key repository contents to perform TLS operations, IBM MQ must be able to retrieve the key repository password.

"Working with SSL/TLS" on page 270
These topics give instructions for performing single tasks related to using TLS with IBM MQ.

ALW *When changes to certificates or the key repository become effective on AIX, Linux, and Windows*

When you change the certificates in a key repository, or the location of the key repository, the changes take effect at a time that depends on the type of channel and how the channel is running.

Changes to the certificates in the key repository, or to the location of the key repository, become effective in the following situations:

- When a new outbound single channel process first runs a TLS channel.
- When a new inbound TCP/IP single channel process first receives a request to start a TLS channel.
- When the MQSC command **REFRESH SECURITY TYPE(SSL)** is issued to refresh the TLS environment.
- For client application processes, when the last TLS connection in the process is closed. The next TLS connection will pick up the certificate changes.
- For channels that run as threads of a process pooling process (amqrmppa), when the process pooling process is started or restarted and first runs a TLS channel. If the process pooling process has already run a TLS channel, and you want the change to become effective immediately, run the MQSC command **REFRESH SECURITY TYPE(SSL)**.
- For channels that run as threads of the channel initiator, when the channel initiator is started or restarted and first runs a TLS channel. If the channel initiator process has already run a TLS channel, and you want the change to become effective immediately, run the MQSC command **REFRESH SECURITY TYPE(SSL)**.
- For channels that run as threads of a TCP/IP listener, when the listener is started or restarted and first receives a request to start a TLS channel. If the listener has already run a TLS channel, and you want the change to become effective immediately, run the MQSC command **REFRESH SECURITY TYPE(SSL)**.

You can also refresh the IBM MQ TLS environment by using the IBM MQ Explorer or PCF commands.

Important: Changes to the keystore configuration file, or to the keystore that is used by an Advanced Message Security (AMS) MCA interceptor or an AMS client, take effect when the queue manger or application is restarted.

ALW *Configuring for cryptographic hardware on AIX, Linux, and Windows*

You can configure cryptographic hardware for a queue manager or client in a number of ways.

You can configure cryptographic hardware for a queue manager on AIX, Linux, and Windows using either of the following methods:

- Use the **ALTER QMGR** MQSC command with the SSLCRYP parameter, as described in ALTER QMGR.
- Use IBM MQ Explorer to configure the cryptographic hardware on your AIX, Linux, and Windows system. For more information, refer to the online help.

You can configure cryptographic hardware for an IBM MQ client on AIX, Linux, and Windows using one of the following methods:

- Set the **MQSSLCRYP** environment variable. The permitted values for **MQSSLCRYP** are the same as for the **SSLCRYP** parameter, as described in ALTER QMGR. To set this environment variable, use one of these commands:
  - Linux AIX On AIX and Linux systems:

    ```
    export MQSSLCRYP=string
    ```

  - Windows On Windows systems:

    ```
    SET MQSSLCRYP=string
    ```

  where *string* represents the parameter string to be used to configure the cryptographic hardware present on the system.

  If you use the GSK_PKCS11 version of the **SSLCRYP** parameter, the PKCS #11 token label must match the label that you configured your hardware with.
- Set the **SSLCryptoHardware** attribute in the SSL stanza of the IBM MQ client configuration file. The permitted values are the same as for the **SSLCRYP** parameter, as described in **ALTER QMGR**.

  If you use the GSK_PKCS11 version of the **SSLCRYP** parameter, the PKCS #11 token label must match the label that you configured your hardware with.

- Set the **CryptoHardware** field of the SSL configuration options structure, MQSCO, on an MQCONNX call. For more information, see Overview for MQSCO.

⚠ **Attention:** >When supplying configuration for the cryptographic hardware through the **MQSSLCRYP** environment variable, or the **SSLCryptoHardware** attribute, you should protect the password prior to storing. For more information, see "IBM MQ clients that use cryptographic hardware" on page 555.

If you have configured cryptographic hardware which uses the PKCS #11 interface using any of these methods, you must store the personal certificate for use on your channels in the key database file for the cryptographic token you have configured. This is described in "Managing certificates on PKCS #11 hardware" on page 548.

MQ Appliance Working with SSL/TLS on IBM MQ Appliance

IBM MQ Appliance has Transport Layer Security (TLS) support.

The IBM MQ Appliance has distinct commands for managing certificates. For detailed information about certificate management, see the IBM MQ Appliance documentation, TLS certificate management

z/OS Working with SSL/TLS on z/OS

This information describes how you set up and work with Transport Layer Security (TLS) on z/OS.

Each topic includes examples of performing each task using RACF. You can perform similar tasks using the other external security managers.

On z/OS, you must also set the number of server subtasks that each queue manager uses for processing TLS calls, as described in "Setting the SSLTASKS parameter on z/OS" on page 302.

z/OS TLS support is integral to the operating system, and is known as *System SSL*. System SSL is part of the Cryptographic Services Base element of z/OS. The Cryptographic Services Base members are installed in the *pdsname*. SIEALNKE partitioned data set (PDS). When you install System SSL, ensure that you choose the appropriate options to provide the CipherSpecs that you require.

If you need to renew a self-signed certificate, see Steps for renewing a self-signed certificate in RACF for more information.

z/OS *Additional user ID requirements for TLS on z/OS*

This information describes the additional requirements your user ID needs to set up and work with TLS on z/OS.

Ensure that you have all the appropriate High Impact or Pervasive (HIPER) updates on your system.

If the key repository is owned by the CHINIT user ID, this user ID needs read access to the IRR.DIGTCERT.LISTRING profile in the FACILITY class, and update access otherwise, and read access to the IRR.DIGTCERT.LIST profile. Grant access by using the PERMIT command with ACCESS(UPDATE) or ACCESS(READ) as appropriate.

Ensure that you have set up the following prerequisites:

- The *ssidCHIN* user ID is defined correctly in RACF, and that the *ssidCHIN* user ID has the appropriate access to the following profiles.

  - IRR.DIGTCERT.LIST
  - IRR.DIGTCERT.LISTRING

  These variables are defined in the RACF FACILITY Class.
- The *ssidCHIN* user ID is the owner of the key ring.
- The personal certificate of the queue manager, if created by the RACDCERT command, is created with a certificate type user ID that is also the same as the *ssidCHIN* user ID.

- The channel initiator is recycled, or the command **REFRESH SECURITY TYPE(SSL)** is issued, to pick up any changes you make to the key ring.
- The IBM MQ Channel Initiator procedure has access to the system SSL runtime library *pdsname*.SIEALNKE through the link list, LPA, or a STEPLIB DD statement. This library must be APF-authorized.
- The user ID under whose authority the channel initiator is running is configured to use z/OS UNIX System Services (z/OS UNIX), as described in the z/OS UNIX System Services Planning documentation.

  Users who do not want the channel initiator to invoke z/OS UNIX using the guest/default UID and OMVS segment, need only model a new OMVS segment based on the default segment as the channel initiator requires no special permissions, and does not run within UNIX as a superuser.

  See the PERMIT commands in “Giving the channel initiator the correct access rights on z/OS” on page 303 for some examples on how you give the channel initiator the correct access.

z/OS *Setting the SSLTASKS parameter on z/OS*

Use the ALTER QMGR command to set the number of server subtasks for processing TLS calls

To use TLS channels, ensure that there are at least two server subtasks by setting the SSLTASKS parameter, using the ALTER QMGR command. For example:

```
ALTER QMGR SSLTASKS(5)
```

To avoid problems with storage allocation, do not set the SSLTASKS attribute to a value greater than eight in an environment where there is no Certificate Revocation List (CRL) checking.

If CRL checking is used, an SSLTASK is held by the channel concerned for the duration of that check. This could be for a significant elapsed time while the relevant LDAP server is contacted, because each SSLTASK is a z/OS task control block.

You must restart the channel initiator if you change the value of the SSLTASKS attribute.

z/OS *Setting up a key repository on z/OS*

Set up a key repository at both ends of the connection. Associate each key repository with its queue manager.

A TLS connection requires a *key repository* at each end of the connection. Each queue manager must have access to a key repository. Use the SSLKEYR parameter on the ALTER QMGR command to associate a key repository with a queue manager. See “The SSL/TLS key repository” on page 25 for more information.

On z/OS, digital certificates are stored in a *key ring* that is managed by your External Security Manager (ESM) . These digital certificates have labels, which associate the certificate with a queue manager. TLS uses these certificates for authentication purposes. All the examples that follow use RACF commands. Equivalent commands exist for other ESM programs.

On z/OS, IBM MQ uses either the value of the **CERTLABL** attribute, if it is set, or the default `ibmWebSphereMQ` with the name of the queue manager appended. See Digital certificate labels for details.

The key repository name for a queue manager is the name of a key ring in your RACF database. You can specify the key ring name either before or after creating the key ring.

Use the following procedure to create a new key ring for a queue manager:

1. Ensure that you have the appropriate authority to issue the RACDCERT command (see Controlling the use of the RACDCERT command for more details).
2. Issue the following command:

   ```
   RACDCERT ID( userid1 ) ADDRING( ring-name )
   ```

   where:

- *userid1* is the user ID of the channel initiator address space, or the user ID that is going to own the key ring (if the key ring is shared).
- *ring-name* is the name you want to give to your key ring. The length of this name can be up to 237 characters. This name is case-sensitive. Specify *ring-name* in uppercase characters to avoid problems.

z/OS *Making CA certificates available to a queue manager on z/OS*

After you have created your key ring, connect any relevant CA certificates to it.

If you have the CA certificate in a data set, you must first add the certificate to the RACF database by using the following command:

```
RACDCERT ID( userid1 ) ADD( input-data-set-name ) WITHLABEL( 'My CA' )
```

Then to connect a CA certificate for `My CA` to your key ring, use the following command:

```
RACDCERT ID(userid1)
CONNECT(CERTAUTH LABEL('My CA') RING(ring-name) USAGE(CERTAUTH))
```

where *userid1* is either the channel initiator user ID or the owner of a shared key ring.

For more information about CA certificates, refer to "Digital certificates" on page 13.

z/OS *Locating the key repository for a queue manager on z/OS*

Use this procedure to obtain the location of your queue manager's key ring.

1. Display your queue manager's attributes, using either of the following MQSC commands:

   ```
   DISPLAY QMGR ALL
   DISPLAY QMGR SSLKEYR
   ```

2. Examine the command output for the location of the key ring.

z/OS *Specifying the key repository location for a queue manager on z/OS*

To specify the location of your queue manager's key ring, use the ALTER QMGR MQSC command to set your queue manager's key repository attribute.

For example:

```
ALTER QMGR SSLKEYR(CSQ1RING)
```

if the key ring is owned by the channel initiator address space, or:

```
ALTER QMGR SSLKEYR(userid1/CSQ1RING)
```

if it is a shared key ring, where *userid1* is the user ID that owns the key ring.

z/OS *Giving the channel initiator the correct access rights on z/OS*

The channel initiator (CHINIT) needs access to the key repository and to certain security profiles.

Granting the CHINIT access to read the key repository

If the key repository is owned by the CHINIT user ID, this user ID needs read access to the IRR.DIGTCERT.LISTRING profile in the FACILITY class, and update access otherwise, and read access to the IRR.DIGTCERT.LIST profile. Grant access by using the PERMIT command with ACCESS(UPDATE) or ACCESS(READ) as appropriate:

```
PERMIT IRR.DIGTCERT.LISTRING CLASS(FACILITY) ID( userid ) ACCESS(UPDATE)
PERMIT IRR.DIGTCERT.LIST CLASS(FACILITY) ID( userid ) ACCESS(READ)
```

where *userid* is the user ID of the channel initiator address space.

Granting the CHINIT read access to the appropriate CSF* profiles

For hardware support provided through the Integrated Cryptographic Service Facility (ICSF) to be used, ensure your CHINIT user ID has read access to the appropriate CSF* profiles in the CSFSERV class by using the following command:

```
PERMIT csf-resource CLASS(CSFSERV) ID( userid ) ACCESS(READ)
```

where *csf-resource* is the name of the CSF* profile and *userid* is the user ID of the channel initiator address space.

Repeat this command for each of the following CSF* profiles:

- CSFDSG
- CSFDSV
- CSFPKD
- CSFPKE
- CSFPKI

Your CHINIT user ID might also need read access to other CSF* profiles. For example, if you are using the ECDHE_RSA_AES_256_GCM_SHA384 Cipher Spec, your CHINIT user ID also needs read access to the following CSF* profiles:

- CSF1DVK
- CSF1GAV
- CSF1GKP
- CSF1SKE
- CSF1TRC
- CSF1TRD

For more information, see RACF CSFSERV resource requirements.

If your certificate keys are stored in ICSF and your installation has established access control over keys stored in ICSF, ensure your CHINIT user ID has read access to the profile in the CSFKEYS class by using the following command:

```
PERMIT IRR.DIGTCERT. userid.* CLASS(CSFKEYS) ID( userid ) ACCESS(READ)
```

where *userid* is the user ID of the channel initiator address space.

Using the Integrated Cryptographic Service Facility (ICSF)

The channel initiator can use ICSF to generate a random number when seeding the password protection algorithm to obfuscate passwords flowing over client channels if TLS is not being used.

For further information, see "Using the Integrated Cryptographic Service Facility (ICSF)" on page 258

z/OS *When changes to certificates or the key repository become effective on z/OS*

Changes become effective when the channel initiator starts or the repository is refreshed.

Specifically, changes to the certificates in the key ring and to the key repository attribute become effective on either of the following occasions:

- When the channel initiator is started or restarted.
- When the REFRESH SECURITY TYPE(SSL) command is issued to refresh the contents of the key repository.

z/OS *Creating a self-signed personal certificate on z/OS*

Use this procedure to create a self-signed personal certificate.

1. Generate a certificate and a public and private key pair using the following command:

```
RACDCERT ID(userid2) GENCERT
SUBJECTSDN(CN('common-name')
           T('title')
           OU('organizational-unit')
           O('organization')
           L('locality')
           SP('state-or-province')
           C('country'))
WITHLABEL('label-name')
```

2. Connect the certificate to your key ring using the following command:

```
RACDCERT ID(userid1)
CONNECT(ID(userid2) LABEL('label-name') RING(ring-name) USAGE(PERSONAL))
```

where:

- *userid1* is the user ID of the channel initiator address space or owner of the shared key ring.
- *userid2* is the user ID associated with the certificate and must be the user ID of the channel initiator address space.

  userid1 and *userid2* can be the same ID.
- *ring-name* is the name you gave the key ring in "Setting up a key repository on z/OS" on page 302.
- *label-name* must be either the value of the IBM MQ **CERTLABL** attribute, if it is set, or the default `ibmWebSphereMQ` with the name of the queue manager appended. See Digital certificate labels for details.

z/OS *Requesting a personal certificate on z/OS*

Apply for a personal certificate using RACF.

To apply for a personal certificate, use RACF as follows:

1. Create a self-signed personal certificate, as in "Creating a self-signed personal certificate on z/OS" on page 305. This certificate provides the request with the attribute values for the Distinguished Name.
2. Create a PKCS #10 Base64-encoded certificate request written to a data set, using the following command:

```
RACDCERT ID(userid2) GENREQ(LABEL(' label_name ')) DSN(' output_data_set_name ')
```

   where

   - *userid2* is the user ID associated with the certificate and must be the user ID of the channel initiator address space
   - *label_name* is the label used when creating the self-signed certificate

   See "Digital certificate labels, understanding the requirements" on page 27 for details.
3. Send the data set to a Certificate Authority (CA) to request a new personal certificate.
4. When the signed certificate is returned to you by the Certificate Authority, add the certificate back into the RACF database, using the original label, as described in "Adding personal certificates to a key repository on z/OS" on page 306.

z/OS *Creating a RACF signed personal certificate*

RACF can function as a certificate authority and issue its own CA certificate.

This section uses the term *signer certificate* to denote a CA certificate issued by RACF.

The private key for the signer certificate must be in the RACF database before you carry out the following procedure:

1. Use the following command to generate a personal certificate signed by RACF, using the signer certificate contained in your RACF database:

```
RACDCERT ID(userid2) GENCERT
SUBJECTSDN(CN('common-name')
           T('title')
           OU('organizational-unit')
           O('organization')
           L('locality')
           SP('state-or-province')
           C('country'))
WITHLABEL('label-name')
SIGNWITH(CERTAUTH LABEL('signer-label'))
```

2. Connect the certificate to your key ring using the following command:

```
RACDCERT ID(userid1)
CONNECT(ID(userid2) LABEL('label-name') RING(ring-name) USAGE(PERSONAL))
```

where:

- *userid1* is the user ID of the channel initiator address space or owner of the shared key ring.
- *userid2* is the user ID associated with the certificate and must be the user ID of the channel initiator address space.

  userid1 and *userid2* can be the same ID.
- *ring-name* is the name you gave the key ring in "Setting up a key repository on z/OS" on page 302.
- *label-name* must be either the value of the IBM MQ **CERTLABL** attribute, if it is set, or the default `ibmWebSphereMQ` with the name of the queue manager or queue sharing group appended. See Digital certificate labels for details.
- *signer-label* is the label of your own signer certificate.

z/OS *Adding personal certificates to a key repository on z/OS*

Use this procedure to add or import a personal certificate to a key ring.

After the certificate authority sends you a new personal certificate, add it to the key ring using the following procedure:

1. Add the certificate to the RACF database using the following command:

```
RACDCERT ID( userid2 ) ADD( input-data-set-name ) WITHLABEL(' label-name ')
```

2. Connect the certificate to your key ring using the following command:

```
RACDCERT ID( userid1 )
CONNECT(ID( userid2 ) LABEL(' label-name ') RING( ring-name ) USAGE(PERSONAL))
```

where:

- *userid1* is the user ID of the channel initiator address space or owner of the shared key ring.
- *userid2* is the user ID associated with the certificate and must be the user ID of the channel initiator address space.

- *ring-name* is the name you gave the key ring in "Setting up a key repository on z/OS" on page 302.
- *input-data-set-name* is the name of the data set containing the CA signed certificate. The data set must be cataloged and must not be a PDS or a member of a PDS. The record format (RECFM) expected by RACDCERT is VB. RACDCERT dynamically allocates and opens the data set, and reads the certificate from it as binary data.
- *label-name* is the label name that was used when you created the original request. It must be either the value of the IBM MQ **CERTLABL** attribute, if it is set, or the default `ibmWebSphereMQ` with the name of the queue manager or queue sharing group appended. See Digital certificate labels for details.

z/OS *Exporting a personal certificate from a key repository on z/OS*

Export the certificate using the RACDCERT command.

On the system from which you want to export the certificate, use the following command:

```
RACDCERT ID(userid2) EXPORT(LABEL('label-name'))
DSN(output-data-set-name) FORMAT(CERTB64)
```

where:

- *userid2* is the user ID under which the certificate was added to the key ring.
- *label-name* is the label of the certificate you want to extract.
- *output-data-set-name* is the data set into which the certificate is placed.
- CERTB64 is a DER encoded X.509 certificate that is in Base64 format. You can choose an alternative format, for example:

  CERTDER
  DER encoded X.509 certificate in binary format

  PKCS12B64
  PKCS #12 certificate in Base64 format

  PKCS12DER
  PKCS #12 certificate in binary format

z/OS *Deleting a personal certificate from a key repository on z/OS*

Delete a personal certificate using the RACDCERT command.

Before deleting a personal certificate, you might want to save a copy of it. To copy your personal certificate to a data set before deleting it, follow the procedure in "Exporting a personal certificate from a key repository on z/OS" on page 307. Then use the following command to delete your personal certificate:

```
RACDCERT ID( userid2 ) DELETE(LABEL(' label-name '))
```

where:

- *userid2* is the user ID under which the certificate was added to the key ring.
- *label-name* is the name of the certificate you want to delete.

z/OS *Renaming a personal certificate in a key repository on z/OS*

Rename a certificate using the RACDCERT command.

If you do not want a certificate with a specific label to be found, but do not want to delete it, you can rename it temporarily using the following command:

```
RACDCERT ID( userid2 ) LABEL(' label-name ') NEWLABEL(' new-label-name ')
```

where:

- *userid2* is the user ID under which the certificate was added to the key ring.

- *label-name* is the name of the certificate you want to rename.
- *new-label-name* is the new name of the certificate.

This can be useful when testing TLS client authentication.

z/OS *Associating a user ID with a digital certificate on z/OS*

IBM MQ can use a user ID associated with a RACF certificate as a channel user ID. Associate a user ID with a certificate by installing it under that user ID, or using a Certificate Name Filter.

The method described in this topic is an alternative to the platform-independent method for associating a user ID with a digital certificate, which uses channel authentication records. For more information about channel authentication records, see "Channel authentication records" on page 51.

When an entity at one end of a TLS channel receives a certificate from a remote connection, the entity asks RACF if there is a user ID associated with that certificate. The entity uses that user ID as the channel user ID. If there is no user ID associated with the certificate, the entity uses the user ID under which the channel initiator is running.

Associate a user ID with a certificate in either of the following ways:

- Install that certificate into the RACF database under the user ID with which you want to associate it, as described in "Adding personal certificates to a key repository on z/OS" on page 306.
- Use a Certificate Name Filter (CNF) to map the Distinguished Name of the subject or issuer of the certificate to the user ID, as described in "Setting up a certificate name filter on z/OS" on page 308.

z/OS *Setting up a certificate name filter on z/OS*

Use the RACDCERT command to define a certificate name filter (CNF), which maps a Distinguished Name to a user ID.

Perform the following steps to set up a CNF.

1. Enable CNF functions using the following command. You require update authority on the class DIGTNMAP to do this.

```
SETROPTS CLASSACT(DIGTNMAP) RACLIST(DIGTNMAP)
```

2. Define the CNF. For example:

```
RACDCERT ID(USER1) MAP WITHLABEL('filter1') TRUST
SDNFILTER('O=IBM.C=UK') IDNFILTER('O=ExampleCA.L=Internet')
```

   where USER1 is the user ID to be used when:

   - The DN of the subject has an Organization of `IBM` and a Country of UK.
   - The DN of the issuer has an Organization of `ExampleCA` and a Locality of `Internet`.

3. Refresh the CNF mappings:

```
SETROPTS RACLIST(DIGTNMAP) REFRESH
```

Note:

1. If the actual certificate is stored in the RACF database, the user ID under which it is installed is used in preference to the user ID associated with any CNF. If the certificate is not stored in the RACF database, the user ID associated with the most specific matching CNF is used. Matches of the subject DN are considered more specific than matches of the issuer DN.
2. Changes to CNFs do not apply until you refresh the CNF mappings.
3. A DN matches the DN filter in a CNF only if the DN filter is identical to the *least significant portion* of the DN. The least significant portion of the DN comprises the attributes that are usually listed at the right-most end of the DN, but which appear at the beginning of the certificate.

For example, consider the SDNFILTER `'O=IBM.C=UK'`. A subject DN of `'CN=QM1.O=IBM.C=UK'` matches that filter, but a subject DN of `'CN=QM1.O=IBM.L=Hursley.C=UK'` does not match that filter.

   The least significant portion of some certificates can contain fields that do not match the DN filter. Consider excluding these certificates by specifying a DN pattern in the SSLPEER pattern on the DEFINE CHANNEL command.

4. If the most specific matching CNF is defined to RACF as NOTRUST, the entity uses the user ID under which the channel initiator is running.
5. RACF uses the `'.'` character as a separator. IBM MQ uses either a comma or a semicolon.

You can define CNFs to ensure that the entity never sets the channel user ID to the default, which is the user ID under which the channel initiator is running. For each CA certificate in the key ring associated with the entity, define a CNF with an IDNFILTER that exactly matches the subject DN of that CA certificate. This ensures that all certificates that the entity might use match at least one of these CNFs. This is because all such certificates must either be connected to the key ring associated with the entity, or must be issued by a CA for which a certificate is connected to the key ring associated with the entity.

Refer to the *z/OS Security Server RACF Security Administrator's Guide* for more information about the commands you use to manipulate CNFs.

z/OS *Defining a sender channel and transmission queue on QMA on z/OS*

Use the **DEFINE CHANNEL** and **DEFINE QLOCAL** commands to set up the required objects.

Procedure

On QMA, issue commands like the following example:

```
DEFINE CHANNEL(TO.QMB) CHLTYPE(SDR) TRPTYPE(TCP) CONNAME(QMB.MACH.COM) XMITQ(QMB)
SSLCIPH(TLS_RSA_WITH_AES_128_CBC_SHA256) DESCR('Sender channel using TLS from QMA to QMB')

DEFINE QLOCAL(QMB) USAGE(XMITQ)
```

Results

A sender channel, TO.QMB, and a transmission queue, QMB, are created.

z/OS *Defining a receiver channel on QMB on z/OS*

Use the **DEFINE CHANNEL** command to set up the required object.

Procedure

On QMB, issue a command like the following example:

```
DEFINE CHANNEL(TO.QMB) CHLTYPE(RCVR) TRPTYPE(TCP) SSLCIPH(TLS_RSA_WITH_AES_128_CBC_SHA256)
SSLCAUTH(REQUIRED) DESCR('Receiver channel using TLS to QMB')
```

Results

A receiver channel, TO.QMB, is created.

z/OS *Starting the sender channel on QMA on z/OS*

If necessary, start a listener program and refresh security. Then start the channel using the **START CHANNEL** command.

Procedure

1. Optional: If you have not already done so, start a listener program on QMB.

   The listener program listens for incoming network requests and starts the receiver channel when it is needed. For information about how to start a listener, see Starting a channel listener.
2. Optional: If any SSL/TLS channels have run previously, issue the command `REFRESH SECURITY TYPE(SSL)`.

   This ensures that all the changes made to the key repository are available.
3. Start the channel on QMA, using the command `START CHANNEL(TO.QMB)`.

Results

The sender channel is started.

z/OS *Exchanging self-signed certificates on z/OS*

Exchange the certificates you previously extracted. If you use FTP, use the correct format.

Procedure

Transfer the CA part of the QM1 certificate to the QM2 system and vice versa, for example, by FTP.

If you transfer the certificates using FTP, you must do so in the correct format.

Transfer the following certificate types in *binary* format:

- DER encoded binary X.509
- PKCS #7 (CA certificates)
- PKCS #12 (personal certificates)

Transfer the following certificate types in ASCII format:

- PEM (privacy-enhanced mail)
- Base64 encoded X.509

z/OS *Defining a sender channel and transmission queue on QM1 on z/OS*

Use the **DEFINE CHANNEL** and **DEFINE QLOCAL** commands to set up the required objects.

Procedure

On QM1, issue commands like the following example:

```
DEFINE CHANNEL(QM1.TO.QM2) CHLTYPE(SDR) TRPTYPE(TCP) CONNAME(QM1.MACH.COM) XMITQ(QM2)
SSLCIPH(TLS_RSA_WITH_AES_128_CBC_SHA) DESCR('Sender channel using TLS from QM1 to QM2')

DEFINE QLOCAL(QM2) USAGE(XMITQ)
```

The CipherSpecs at each end of the channel must be the same.

Only the SSLCIPH parameter is mandatory if you want your channel to use TLS. See "CipherSpecs and CipherSuites in IBM MQ" on page 41 for information about the permitted values for the SSLCIPH parameter.

Results

A sender channel, QM1.TO.QM2, and a transmission queue, QM2, are created.

z/OS *Defining a receiver channel on QM2 on z/OS*

Use the **DEFINE CHANNEL** command to set up the required object.

Procedure

On QM2, issue a command like the following example:

```
DEFINE CHANNEL(QM1.TO.QM2) CHLTYPE(RCVR) TRPTYPE(TCP) SSLCIPH(TLS_RSA_WITH_AES_128_CBC_SHA256)
SSLCAUTH(REQUIRED) DESCR('Receiver channel using TLS from QM1 to QM2')
```

The channel must have the same name as the sender channel you defined in "Defining a sender channel and transmission queue on QM1 on z/OS" on page 310, and use the same CipherSpec.

z/OS *Starting the sender channel on QM1 on z/OS*

If necessary, start a listener program and refresh security. Then start the channel using the **START CHANNEL** command.

Procedure

1. Optional: If you have not already done so, start a listener program on QM2.

   The listener program listens for incoming network requests and starts the receiver channel when it is needed. For information about how to start a listener, see Starting a channel listener
2. Optional: If any SSL/TLS channels have run previously, issue the command REFRESH SECURITY TYPE(SSL).

   This ensures that all the changes made to the key repository are available.
3. On QM1, start the channel, using the command `START CHANNEL(QM1.TO.QM2)`.

Results

The sender channel is started.

z/OS *Refreshing the SSL or TLS environment on z/OS*

Refresh the TLS environment on queue manager QMA using the **REFRESH SECURITY** command.

Procedure

On QMA, enter the following command:

```
REFRESH SECURITY TYPE(SSL)
```

This ensures that all the changes made to the key repository are available.

z/OS *Allowing anonymous connections on a receiver channel on z/OS*

Use the **ALTER CHANNEL** command to make SSL or TLS client authentication optional.

Procedure

On QMB, enter the following command:

```
ALTER CHANNEL(TO.QMB) CHLTYPE(RCVR) SSLCAUTH(OPTIONAL)
```

z/OS *Starting the sender channel on QM1 on z/OS*

If necessary, start the channel initiator, start a listener program, and refresh security. Then start the channel using the **START CHANNEL** command.

Procedure

1. Optional: if you have not already done so, start the channel initiator.
2. Optional: If you have not already done so, start a listener program on QM2.

   The listener program listens for incoming network requests and starts the receiver channel when it is needed. For information about how to start a listener, see Starting a channel listener

3. Optional: If the channel initiator was already running or any SSL/TLS channels have run previously, issue the command REFRESH SECURITY TYPE(SSL).

   This ensures that all the changes made to the key repository are available.
4. On QM1, start the channel, using the command `START CHANNEL(QM1.TO.QM2)`.

Results

The sender channel is started.

z/OS *Starting the sender channel on QMA on z/OS*

If necessary, start the channel initiator, start a listener program, and refresh security. Then start the channel using the **`START CHANNEL`** command.

Procedure

1. Optional: If you have not already done so, start the channel initiator.
2. Optional: If you have not already done so, start a listener program on QMB.

   The listener program listens for incoming network requests and starts the receiver channel when it is needed. For information about how to start a listener, see Starting a channel listener.
3. Optional: If the channel initiator was already running or if any SSL/TLS channels have run previously, issue the command `REFRESH SECURITY TYPE(SSL)`.

   This ensures that all the changes made to the key repository are available.
4. Start the channel on QMA, using the command `START CHANNEL(TO.QMB)`.

Results

The sender channel is started.

z/OS *Modifying elliptic curve key length on z/OS*

How you modify the GSK_CLIENT_ECURVE_LIST environment variable, to set the list of elliptic curves or supported groups that are specified by the client, as a string consisting of one or more 4-character values in order of preference for use.

Important: You must apply the fix in z/OS APAR OA61783 to permit certain elliptic curves to be made effective by the operating system, when using TLS 1.0, TLS 1.1 and/or TLS 1.2 negotiated connections.

You can set this TLS environment variable in the channel initiator startup JCL, using the CEEOPTS DD statement:

```
CEEOPTS DD DSN=<dataset-name>,DISP=SHR
```

In the dataset referenced above, specify the list that you want to use, for example:

```
ENVAR("GSK_CLIENT_ECURVE_LIST=002300240025")
```

Important: Do not use this CEEOPTS statement with in-stream data, as this prevents the environment variable from being set for all TLS tasks using that statement.

Ensure you reference a sequential dataset, or partitioned dataset member, to allow this to work when using an SSLTASKS value greater than one.

You can also use the server analogue equivalent of GSK_CLIENT_ECURVE_LIST, which is GSK_SERVER_ALLOWED_KEX_ECURVES. See Limiting key exchange elliptic curves for more information.

In addition, see Table 5 in Cipher suite definitions for a list of valid 4-character elliptic curve and supported groups specifications.

The default specification is `00210023002400250019`. If TLS V1.3 is enabled, `0029 (x25519)` is appended to the end of the default list.

Identifying and authenticating users

You can identify and authenticate users by using X.509 certificates, the MQCSP structure, or in several types of user exit program.

Using X.509 certificates

You can identify and authenticate users by using X.509 certificates with the **SET CHLAUTH** command and **SSLPEER** parameter. The **SSLPEER** parameter specifies a filter to use to compare with the Subject Distinguished Name of the certificate from the peer queue manager or client at the other end of the channel.

For more information about using the **SET CHLAUTH** command and **SSLPEER** parameter, see SET CHLAUTH.

Digital certificates can be revoked by Certificate Authorities. You can check the revocation status of certificates using OCSP, or CRLs on LDAP servers, depending on platform. For more information, see "Working with revoked certificates" on page 332.

Using the MQCSP structure

The MQCSP connection security parameters structure is specified on an MQCONNX call. This structure can contain credentials that are supplied by the application. The application can supply a user ID and password in the MQCSP structure. From IBM MQ 9.3.4, applications can also supply an authentication token. If necessary, the MQCSP can be altered in a security exit.

Warning: The credentials in an MQCSP structure are sometimes sent across the network in plain text. To ensure that client application credentials are protected, see "MQCSP password protection" on page 31.

For more information, see "Identifying and authenticating users using the MQCSP structure" on page 314 and "Working with authentication tokens" on page 318.

Linux AIX On AIX and Linux, the user ID and password that is specified in the MQCSP structure can be authenticated by using either the operating system or Pluggable Authentication Method (PAM). PAM provides a general mechanism for user authentication that hides the details from services. For more information, see "Using the Pluggable Authentication Method (PAM)" on page 343.

Implementing identification and authentication in exits

You can identify and authenticate users by using several types of user exit program. For more information, see "Implementing identification and authentication in security exits" on page 316, "Identity mapping in message exits" on page 317, and "Identity mapping in the API exit and API-crossing exit" on page 317.

Privileged users

A privileged user is one that has full administrative authorities for IBM MQ.

In addition to the users listed in the following table, there are certain objects and authorizations for which extra care must be taken when granting access, to ensure integrity and security of the queue manager. Extra scrutiny must be applied when granting any of the following authorizations:

- Any authorizations to SYSTEM objects
- Administration authorizations to create, alter and delete objects.

  z/OS On z/OS, this authorization is command security and command resource security authority to issue DEFINE, ALTER and DELETE commands.

  Multi On all other platforms, these authorizations are administration authorizations such as `+crt`, `+chg` and `+dlt`.
- Administration authorization to clear queues.

  z/OS On z/OS, this authorization is command security and command resource security authority to issue CLEAR commands.

  Multi On all other platforms, this authorization is `+clr`.

- Administration authorizations to stop channels, backout or commit messages.

  z/OS On z/OS, this authorization is command security and command resource security authority to issue commands such as RESET CHANNEL, START CHANNEL and STOP CHANNEL.

  Multi On all other platforms, these authorizations are `+ctrl` and `+ctrlx`.

- Alternate user MQI authorization that allows applications to escalate privileges for authorization checks.

  z/OS On z/OS, this authorization is any authority granted to the alternate user security profiles.

  Multi On all other platforms, this authorization is `+altusr`.

- Context authorizations that allow applications to change the security context of messages.

  z/OS On z/OS, this authorization is any authority granted to the context security profiles.

  Multi On all other platforms, these authorizations are `+setall` and `+setid`.

As a general principal, messaging applications should only be granted the basic MQI authorizations to the queues or topics that are needed. MCA channels that execute under a non-privileged MCAUSER and certain other special types of applications, such as dead-letter queue handlers may require additional authorizations not normally granted to applications to operate correctly.

Table 67. Privileged users by platform

Platform	**Privileged users**
Windows systems	• SYSTEM • Members of the mqm group • Members of the Administrators group
AIX and Linux systems	• Members of the mqm group
IBM i systems	• The profiles qmqm and qmqmadm • All members of the qmqmadm group • Any user defined with the *ALLOBJ setting
z/OS	The user ID that the channel initiator, queue manager and advanced message security address spaces are running under. These user IDs do not automatically have full administrative authorities for IBM MQ, but are considered privileged due to the level of access that is typically granted to these user IDs.

Identifying and authenticating users using the MQCSP structure

You can specify the MQCSP connection security parameters structure on an MQCONNX call. The MQCSP structure is the primary way for applications that use the message queue interface (MQI) to control the credentials that are used for authentication.

The MQCSP structure contains credentials that the authorization service can use to identify and authenticate the user.

The MQCSP structure can be modified by client or server-side security exits, even if the application does not explicitly provide the MQCSP structure. An example of application that does not explicitly provide an MQCSP structure is an application that uses IBM MQ classes for JMS. For an example of a client-side security exit that inserts a user ID and password in the MQCSP structure, see "Client side security exit to insert user ID and password (mqccred)" on page 80.

V 9.4.0 The MQCSP structure contains a user ID and password, or an authentication token. The following restrictions apply to credentials supplied in the MQCSP structure:

- An application or exit must supply either a user ID and password, or an authentication token, but not both.
- Only authentication tokens that meet specific formats and requirements can be used to access IBM MQ. For more information about the requirements for authentication tokens in IBM MQ, see "Requirements for authentication tokens" on page 321.
- If the identity in the authentication token is to be adopted as the context for the application, the token must provide a suitable user claim, and the claim value must be a valid IBM MQ user ID. For example, the username must comply with maximum length and special character restrictions. For more information about adopting a user ID, see "Relationship between MQCSP and ADOPTCTX settings" on page 315.

For more information about the MQCSP structure, see MQCSP - Security parameters.

Warning: The credentials in an MQCSP structure for a client application are sometimes sent across the network in plain text. To ensure that client application credentials are protected, see "MQCSP password protection" on page 31.

Relationship between MQCSP and ADOPTCTX settings

IBM MQ always authenticates credentials that are passed in the MQCSP structure if the connection authentication feature is enabled. After the credentials are authenticated successfully, IBM MQ can adopt the user ID for subsequent authorization checks on operations performed by the connected application. The user ID in the MQCSP credentials is adopted if the authentication information (AUTHINFO) object that is referenced by the queue manager's **CONNAUTH** attribute is defined with **`ADOPTCTX(YES)`**.

IBM MQ has a limit on the length of user IDs that it can use for authorization checks. For more information about these limits, see "User IDs" on page 88. When a user ID passed in the MQCSP structure is adopted, IBM MQ behaves differently, depending on other configuration options:

- When using LDAP connection authentication, IBM MQ adopts the user ID that is in the short username attribute of the user's LDAP record. The short username attribute is set using the **SHORTUSR** attribute of the AUTHINFO object.

  For example, if **SHORTUSR** is set to `'CN'`, and the LDAP record lists the user as `'CN=Test,SN=MQ,O=IBM,C=UK'`, the user ID `Test` is used.
- When using OS connection authentication or PAM authentication, if ADOPTCTX is YES, the user ID passed in the MQCSP structure is truncated in order to meet the 12 character user ID limit of IBM MQ when adopted as the connection context.

  If **`ChlAuthEarlyAdopt`** is enabled, the truncation happens after the user credentials have been authenticated.

  If **`ChlAuthEarlyAdopt`** is not enabled, the truncation happens before adoption. On Windows, if the user is supplied in the format `user@domain`, this means that the truncation can result in a domain specification that is not valid when the user is less than 12 characters.

  For example if a user `` `ibmmq@windowsdomain` `` is provided through the MQCSP, it is truncated to `` `ibmmq@window` `` in this scenario. This results in the following error:

  ```
  AMQ8074W: Authorization failed as the SID 'SID' does not match the entity 'ibmmq@window'
  ```

  On this basis, if you pass a user ID longer than 12 characters, such as a Windows domain user ID in the form `user@domain`, through the MQCSP you should configure **`ChlAuthEarlyAdopt`**=*Y* in the qm.ini file to avoid this error.

Alternatively, use ADOPTCTX(NO) on the CONNAUTH AUTHINFO configuration, and use an alternate approach such as a CHLAUTH USERMAP rule, a security exit, or the channel object MCAUSER setting to set the user ID for the channel.

Implementing identification and authentication in security exits

You can use a security exit to implement one-way or mutual authentication.

The primary purpose of a security exit is to enable the MCA at each end of a channel to authenticate its partner. At each end of a message channel, and at the server end of an MQI channel, an MCA typically acts on behalf of the queue manager to which it is connected. At the client end of an MQI channel, an MCA typically acts on behalf of the user of the IBM MQ MQI client application. In this situation, mutual authentication actually takes place between two queue managers, or between a queue manager and the user of an IBM MQ MQI client application.

The supplied security exit (the SSPI channel exit) illustrates how mutual authentication can be implemented by exchanging authentication tokens that are generated, and then checked, by a trusted authentication server such as Kerberos. For more details, see "The SSPI channel exit program on Windows" on page 154.

Mutual authentication can also be implemented by using Public Key Infrastructure (PKI) technology. Each security exit generates some random data, signs it using the private key of the queue manager or user it is representing, and sends the signed data to its partner in a security message. The partner security exit performs the authentication by checking the digital signature using the public key of the queue manager or user. Before exchanging digital signatures, the security exits might need to agree the algorithm for generating a message digest, if more than one algorithm is available for use.

When a security exit sends the signed data to its partner, it also needs to send some means of identifying the queue manager or user it is representing. This might be a Distinguished Name, or even a digital certificate. If a digital certificate is sent, the partner security exit can validate the certificate by working through the certificate chain to the root CA certificate. This provides assurance of the ownership of the public key that is used to check the digital signature.

The partner security exit can validate a digital certificate only if it has access to a key repository that contains the remaining certificates in the certificate chain. If a digital certificate for the queue manager or user is not sent, one must be available in the key repository to which the partner security exit has access. The partner security exit cannot check the digital signature unless it can find the signer's public key.

Transport Layer Security (TLS) uses PKI techniques like the ones just described. For more information about how the Secure Sockets Layer performs authentication, see "Transport Layer Security (TLS) concepts" on page 18.

If a trusted authentication server or PKI support is not available, other techniques can be used. A common technique, which can be implemented in security exits, uses a symmetric key algorithm.

One of the security exits, exit A, generates a random number and sends it in a security message to its partner security exit, exit B. Exit B encrypts the number using its copy of a key which is known only to the two security exits. Exit B sends the encrypted number to exit A in a security message with a second random number that exit B has generated. Exit A verifies that the first random number has been encrypted correctly, encrypts the second random number using its copy of the key, and sends the encrypted number to exit B in a security message. Exit B then verifies that the second random number has been encrypted correctly. During this exchange, if either security exit is not satisfied with the authenticity of other, it can instruct the MCA to close the channel.

An advantage of this technique is that no key or password is sent over the communications connection during the exchange. A disadvantage is that it does not provide a solution to the problem of how to distribute the shared key in a secure way. One solution to this problem is described in "Implementing confidentiality in user exit programs" on page 456. A similar technique is used in SNA for the mutual authentication of two LUs when they bind to form a session. The technique is described in "Session level authentication" on page 121.

All the preceding techniques for mutual authentication can be adapted to provide one-way authentication.

Identity mapping in message exits

You can use message exits to process information to authenticate a user ID, though it might be better to implement authentication at the application level.

When an application puts a message on a queue, the *UserIdentifier* field in the message descriptor contains a user ID associated with the application. However, there is no data present that can be used to authenticate the user ID. This data can be added by a message exit at the sending end of a channel and checked by a message exit at the receiving end of the channel. The authenticating data can be an encrypted password or a digital signature, for example.

This service might be more effective if it is implemented at the application level. The basic requirement is for the user of the application that receives the message to be able to identify and authenticate the user of the application that sent the message. It is therefore natural to consider implementing this service at the application level. For more information, see "Identity mapping in the API exit and API-crossing exit" on page 317.

Identity mapping in the API exit and API-crossing exit

An application that receives a message must be able to identify and authenticate the user of the application that sent the message. This service is typically best implemented at the application level. API exits can implement the service in a number of ways.

At the level of an individual message, identification and authentication is a service that involves two users, the sender and the receiver of the message. The basic requirement is for the user of the application that receives the message to be able to identify and authenticate the user of the application that sent the message. Note that the requirement is for one way, not two way, authentication.

Depending on how it is implemented, the users and their applications might need to interface, or even interact, with the service. In addition, when and how the service is used might depend on where the users and their applications are located, and on the nature of the applications themselves. It is therefore natural to consider implementing the service at the application level rather than at the link level.

If you consider implementing this service at the link level, you might need to resolve issues such as the following:

- On a message channel, how do you apply the service only to those messages that require it?
- How do you enable users and their applications to interface, or interact, with the service, if this is a requirement?
- In a multi-hop situation, where a message is sent over more than one message channel on the way to its destination, where do you invoke the components of the service?

Here are some examples of how the identification and authentication service can be implemented at the application level. The term *API exit* means either an API exit or an API-crossing exit.

- When an application puts a message on a queue, an API exit can acquire an authentication token from a trusted authentication server such as Kerberos. The API exit can add this token to the application data in the message. When the message is retrieved by the receiving application, a second API exit can ask the authentication server to authenticate the sender by checking the token.
- When an application puts a message on a queue, an API exit can append the following items to the application data in the message:
  - The digital certificate of the sender
  - The digital signature of the sender

  If different algorithms for generating a message digest are available for use, the API exit can include the name of the algorithm it has used.

  When the message is retrieved by the receiving application, a second API exit can perform the following checks:

  - The API exit can validate the digital certificate by working through the certificate chain to the root CA certificate. To do this, the API exit must have access to a key repository that contains the remaining

  certificates in the certificate chain. This check provide assurance that the sender, identified by the Distinguished Name, is the genuine owner of the public key contained in the certificate.
  - The API exit can check the digital signature using the public key contained in the certificate. This check authenticates the sender.

  The Distinguished Name of the sender can be sent instead of the whole digital certificate. In this case, the key repository must contain the sender's certificate so that the second API exit can find the public key of the sender. Another possibility is to send all the certificates in the certificate chain.

- When an application puts a message on a queue, the *UserIdentifier* field in the message descriptor contains a user ID associated with the application. The user ID can be used to identify the sender. To enable authentication, an API exit can append some data, such as an encrypted password, to the application data in the message. When the message is retrieved by the receiving application, a second API exit can authenticate the user ID by using the data that has travelled with the message.

  This technique might be considered sufficient for messages that originate in a controlled and trusted environment, and in circumstances where a trusted authentication server or PKI support is not available.

Linux V 9.4.0 AIX Working with authentication tokens

From IBM MQ 9.4.0, client applications can provide tokens to authenticate with a queue manager running on AIX or Linux. The user ID in the token can also be used for authorization to access IBM MQ resources.

JWTs (JSON Web Tokens) adopt a claims-based identity model. The identity and access control are abstracted into ideas of claims and token issuers.

- A claim is a name value pair that contains information about a user and establishes who the user is, not what they can do.
- The token issuer is a trusted third party or a server that is issues a token for a user based only on the identity of the user. The token issuer is not concerned with what the user can do.

A token is a simple structure that contains claims and can easily be transferred between parties over the internet. Using tokens for authentication has the benefit of centralized identity management. You can use one trusted token issuer so your applications can authenticate with many services without separately registering with each service. Tokens provide increased security as credentials are not sent to each service, only to the trusted issuer.

A JWT is defined through the proposed internet standard RFC7519.

How tokens work with IBM MQ

Tokens that are used with IBM MQ must be valid JWTs that have been signed with an algorithm that IBM MQ supports. The JWT must be signed according to the JSON Web Signature (JWS) standard. Tokens that use JSON Web Encryption (JWE) and JSON Web Key (JWK) JOSE technologies cannot be used with IBM MQ. For more information, see "Requirements for authentication tokens" on page 321.

The application that supplies the authentication token can run on any platform that supports IBM MQ clients. The application must be written in C or in Java, and connect to the queue manager using client bindings. However, the queue manager must run on AIX or Linux.

The queue manager validates the token signature against the trusted issuer public key or symmetric key in the key repository. To set up the queue manager, follow the steps in "Configuring a queue manager to accept authentication tokens using a JWKS endpoint" on page 324 or Configuring a queue manager to accept authentication tokens using a local keystore.

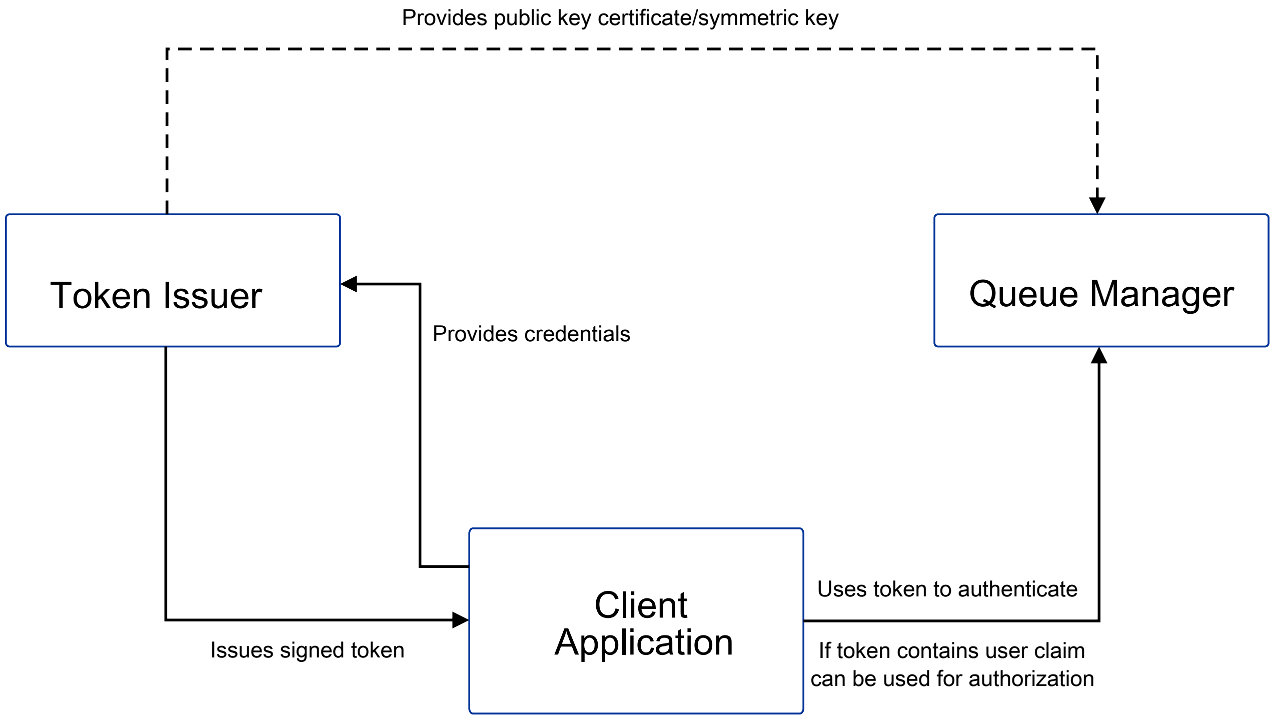
The token issuer is the trusted party that has the delegated security access, meaning they verify the identity of the application user. The queue manager checks that an authentication token is valid and that the authenticated user is authorized to access IBM MQ objects. The queue manager can, but does not need to know of the users before they first connect in with a token. The IBM MQ administrator must set up authentication and authorization for the applications that connect to the queue manager, and set the requirements for what the tokens must contain.

The client application can dynamically request a token from the issuer that it uses for authentication when it connects to IBM MQ. The application then uses the MQCSP structure, or the equivalent in the chosen API, to pass the token to the queue manager when it connects.

If the application cannot be changed to request an authentication token and present the token to the queue manger when it connects, a security exit can alternatively be used to provide a token in the MQCSP structure.

If the token meets the requirements for authentication tokens, and token signature is valid, the connection is established. The queue manager can also use the user ID contained in the token for authorization checks to access IBM MQ resources if the optional user claim is contained in the token. The user claim is the claim within the token that contains the user ID that the queue manager adopts for authorization checks. This name of the user claim is specified with the **`UserClaim`** attribute in the **`AuthToken`** stanza of the `qm.ini` file.

For more information, see "Using authentication tokens in an application" on page 329 and MQCSP - Security parameters.


Provides public key certificate/symmetric key
Token Issuer
Queue Manager
Provides credentials
Client Application
Uses token to authenticate
Issues signed token
If token contains user claim can be used for authorization

The diagram shows a basic example of the expected flow for use of tokens with IBM MQ. The expected lifecycle is as follows:

- The token is issued to an application by the trusted issuer. For more information, see Requirements for authentication tokens.
- The application passes the token into the queue manager when connecting. For more information, see Using authentication tokens in an application.
- The queue manager validates the token signature against the trusted issuer public key or symmetric key in the key repository. To set up the queue manager, follow the steps in "Configuring a queue manager to accept authentication tokens using a JWKS endpoint" on page 324.
- If the authentication token contains a valid user claim, the user in the token can be adopted for authorization checks to access IBM MQ resources . For more information, see Adopting users for authorization.
- The IBM MQ administrator manages trusted token issuer certificates. When the certificate expires, a new certificate must be obtained from the token issuer and added to the key repository.
- If you configured your queue manager and the application is connecting but encounter issues with the token, see Troubleshooting authentication token problems and Token authentication error codes.

IBM MQ works with any token issuer that provides tokens that conform to the JWT and JWS standards.

If you are not already using tokens but want to understand what is involved in standing up a token server, see the Getting started guide for the free and open source Keycloak project.

Related reference
AuthToken stanza of the `qm.ini` file

Linux V 9.4.0 AIX Requirements for authentication tokens

Validation requirements, structure, and algorithms for authentication tokens used with IBM MQ.

Requirements

Authentication tokens that are used with IBM MQ must meet the following requirements.

- The token length must not exceed the maximum length of 8192 characters. For more information, see TokenLength (MQLONG) for MQCSP.
- The token structure and encoding is valid as defined by the JSON Web Token (JWT) specification in RFC7519, and the JSON Web Signature (JWS) specification in RFC7515.
- The required token header parameters that are specified in Table 68 on page 322 are present and the values of the parameters are valid.
- The required payload claims specified in Table 69 on page 323 are present and the values of the claims are valid.
- The token is signed with an algorithm in Table 70 on page 323 that IBM MQ supports.
- The value of the expiry (**exp**) claim is later than the current time.
- If the not before (**nbf**) claim is present, the value is before the current time.
- If a user claim is present, the value must meet the requirements for "User IDs in authentication tokens" on page 324.

Token structure

IBM MQ accepts JWTs that conform to the RFC7519 standard. The JWT must be signed and encoded according to the JWS standard that is defined in RFC7515.

IBM MQ expects the JWS secured token to contain the following three components:

JOSE header

A JSON object that contains parameters that describe the type of token and the cryptographic algorithms that are used to secure its contents.

The following header example declares that the encoded object is a JWT, and that the header and the payload are secured by using the HMAC SHA-256 algorithm.

```
{
 "typ":"JWT",
 "alg":"HS256"
}
```

JWS payload

A JSON object that contains claims as specified in the JWT standard. Each member of the JSON object is a claim. Claims can assert the identity of the token issuer, or the user ID of the bearer.

```
{
 "exp": 1685529153,
 "nbf": 1685528150,
 "AppUser": "MyUserName"
 }
```

JWS signature

Used to validate that the token is issued by a trusted issuer.

These components are represented in the JWS secured token as base64url-encoded strings separated by a period ('.').

An authentication token that conforms to the JWS standard is signed to allow the token's authenticity to be validated, but it is not encrypted. Therefore, it can be read, and possibly reused, by anyone who has access to the token. Configure the connection to the queue manager to ensure that the authentication is protected by using encryption when it is sent over the network, for example, by using TLS. For more information about the options to protect credentials that are supplied by an application, see MQCSP password protection.

IBM MQ supports the following parameters and claims in the header and the payload of authentication tokens. Any additional parameters or claims in a token are ignored. If a token contains more than one parameter or claim with the same name, the last parameter or claim with the duplicate name is used.

Table 68. Token header parameter descriptions

Token part	Parameter name	Data type	Required	Description
Header	**typ**	String	Yes	The token type. The value of this parameter must be "JWT".
	alg	String	Yes	The algorithm used to secure the header and the payload. The value of this parameter must be one of the algorithms in Table 70 on page 323.

Table 69. Token payload claims descriptions				
Token part	**Parameter name**	**Data type**	**Required**	**Description**
Payload	**exp**	Integer	Yes	The token expiry time, expressed as the number of seconds since 1 January 1979, 00:00 Coordinated Universal Time. The token is not accepted after this time.
	nbf	Integer	No	The time, expressed as the number of seconds since 1 January 1979, 00:00 Coordinated Universal Time before which the token is not accepted.
	User claim name specified the in the **UserClaim** field of the **AuthToken** stanza in the `qm.ini` file.	String	Required only if the user claim in the token is used for authorization.	The name of the claim that contains the user ID that is adopted for authorization checks. For example, if your token has the user claim `"AppUser": "MyUserName"`, then you must specify **UserClaim**=`AppUser` in the **AuthToken** stanza of the `qm.ini` file.

For a good example of an encoded and decoded token, see the debugger page on the `jwt.io` website.

Algorithms

IBM MQ supports a subset of algorithms that are included in the JSON Web Algorithms (JWA) specification for JWS secured tokens.

Table 70. JSON Web Algorithms (JWA) supported by IBM MQ for JWS secured tokens	
`alg` parameter value	**Digital Signature or MAC Algorithm**
HS256	HMAC using SHA-256
HS384	HMAC using SHA-384
HS512	HMAC using SHA-512
RS256	RSASSA-PKCS1-v1_5 using SHA-256
RS384	RSASSA-PKCS1-v1_5 using SHA-384
RS512	RSASSA-PKCS1-v1_5 using SHA-512

Asymmetric key certificate requirements

If a token is signed with an asymmetric key, the public key certificate from the token issuer must be in the key repository that the queue manager uses for token authentication. When the authentication token is received, the certificate must be within its validity period. No checks are made to ensure that the certificate from the token issuer has not been revoked.

User IDs in authentication tokens

If the queue manager is configured to adopt the user ID that is contained in the user claim of an authentication token as the context for the application, the user ID that is adopted must meet the following requirements:

- It can contain up to 12 characters.
- It must start with one of the following characters:

  A-Z a-z
- It can contain any of the following characters:

  0-9 A-Z a-z + , - . : = _
- It must not be one of the reserved user IDs UNKNOWN and NOBODY.

Related tasks

Configuring a queue manager to accept **AuthTokens**

Related reference

AuthToken stanza of the `qm.ini` file

Linux V 9.4.0 AIX Configuring a queue manager to accept authentication tokens using a JWKS endpoint

Configure your IBM MQ queue manager running on AIX or Linux to authenticate users and applications with authentication tokens using a JWKS endpoint.

Before you begin

For more information about how tokens work with IBM MQ, see Working with authentication tokens.

Before you configure your queue manager, check that the AUTHINFO object that is referenced in the queue manager **CONNAUTH** attribute is of type `IDPWOS`. Token authentication is only available when the queue manager is configured for OS user ID and password checking.

Check that the **SecurityPolicy** attribute of the Service stanza is not set to `Group`. Token authentication is not available if **SecurityPolicy** is explicitly set to `Group`. If **SecurityPolicy** is set to `Group`, remove the **SecurityPolicy** attribute from the Service stanza, then restart the queue manager.

About this task

Applications can authenticate with the queue manager by using tokens. IBM MQ accepts JSON Web Tokens (*JWTs*) from trusted issuers that follow the proposed internet standard RFC7519. You can use tokens to authenticate an identity, which then can be adopted for future authorization checks.

The simplest way to configure your queue manager to accept tokens is to point to a JWKS endpoint as described below. If your authentication service does not provide such and endpoint or JWKS is unsuitable for other reasons, see "Configuring a queue manager to accept authentication tokens using a local keystore" on page 325.

Procedure

1. Ask your authentication server administrator for these details:
   - The correct JWKS endpoint (URL).
   - What certificate this server uses to encrypt HTTP traffic and/or which authority signs this certificate.

     Important: You should always provide JWKS information over TLS/HTTPS and you need this information to ensure that the queue manager can trust the connection.
2. Configure the queue manager to create outgoing https connections by providing an **HTTPSKeyStore** in the qm.ini file.

For more information, see

- The HTTPSKeyStore explanation in the `qm.ini` file.
- "Creating a key repository for use as a TLS truststore" on page 331.

If the authentication server uses a bespoke certificate/CA then you need to ensure this is correctly present in this HTTPSKeyStore.

3. Configure the JWKS endpoint by defining a JWKS stanza in the qm.ini configuration file.

   The additional stanza provides the following:

   - **`issuername`**. This must match the '`iss`' claim that is present in any tokens signed by this authority, and is often based on the URL of the authentication service.
   - **`endpoint`**. This is the address from which the queue manager queries public keys used to validate token signatures.
   - **`userclaim`**.This is optional for identifying a custom field in tokens which should be used for IBM MQ authority checks once a token has been validated.

   ⚠ **Attention:** This must be present if you intend to use **ADOPTCTX**(YES) for such connections.

4. Once the `.ini` file changes are complete, issue the command `REFRESH SECURITY TYPE(AUTHINFO)` or restart the queue manager.

   If the configuration is successful, applications are able to connect using signed tokens immediately.

   If there are any problems, for example, unable to contact the authentication service to retrieve public keys, the problems are reported in the AMQERR01 log file for the queue manager.

Results

You have successfully configured a queue manager to accept authentication tokens using a JWKS endpoint.

Note: Keys are periodically refreshed from the authentication server (every 15 minutes), and more frequently if an unknown key ID is presented by a connecting application. Typically, this means that no further IBM MQ configuration actions are required to update certificates as they expire and are replaced on the server side. To force an immediate refresh, issue the command `REFRESH SECURITY TYPE(AUTHINFO)` at any time.

Related concepts

Troubleshooting authentication token problems

Related tasks

Using authentication tokens in an application

Related reference

AuthToken stanza of the `qm.ini` file

Linux V 9.4.0 AIX Configuring a queue manager to accept authentication tokens using a local keystore

Configure your IBM MQ queue manager to authenticate users and applications with authentication tokens.

Before you begin

Where possible, consider using a JWKS endpoint, see "Configuring a queue manager to accept authentication tokens using a JWKS endpoint" on page 324, rather than manually configuring your token validation certificates. Using JWKS typically makes both initial configuration and ongoing maintenance simpler.

Read about how tokens work with IBM MQ in Working with authentication tokens.

Before you configure your queue manager, check that the AUTHINFO object that is referenced in the queue manager **CONNAUTH** attribute is of type IDPWOS. Token authentication is only available when the queue manager is configured for OS user ID and password checking.

Check that the **SecurityPolicy** attribute of the Service stanza is not set to Group. Token authentication is not available if **SecurityPolicy** is explicitly set to Group. If **SecurityPolicy** is set to Group, remove the **SecurityPolicy** attribute from the Service stanza, then restart the queue manager.

About this task

From IBM MQ 9.3.4 applications can authenticate with the queue manager by using tokens. IBM MQ accepts JSON Web Tokens (*JWTs*) from trusted issuers that follow the proposed internet standard RFC7519. You can use tokens to authenticate an identity, which then can be adopted for future authorization checks.

Configure your queue manager to accept tokens by saving the trusted issuer's public key certificate or symmetric key to the queue manager's key repository. Add the AuthToken stanza to the `qm.ini` file and refresh the security configuration so the queue manager picks up the new configuration.

You might want to configure a local keystore rather than using JWKS in a test environment, or when direct connectivity to your authentication server from your queue manager is not possible. You can also define a local keystore in addition to any JWKS endpoints.

Note: Where both a JWKS endpoint and a local keystore provide a matching issuer and KID for a presented token, the JWKS endpoint supplied key is used in preference.

In these situations, configure the local keystore as follows:

Procedure

1. Create the key repository.

   a) Create a key repository for the public key certificate or symmetric key that is received from the trusted issuer. You can use either a CMS key repository with the file extension `.kdb` or a PKCS#12 key repository with the file extension `.p12`.

   Issue the following command to create a CMS key repository:

   ```
   runmqakm -keydb -create -db /var/mqm/qmgrs/qm1/tokenissuer/key.kdb -pw MyKeystorePassword
   -type cms
   ```

   If the **runmqakm** command returns an error, see runmqakm -keydb. If the command completes successfully, use the `ls` command to list the contents of the directory:

   ```
   ls -l /var/mqm/qmgrs/qm1/tokenissuer
   ```

   The following files are displayed:

   ```
   -rw------- 1 adminuser mqm 88 Feb 22 07:50 key.crl
   -rw------- 1 adminuser mqm 88 Feb 22 07:50 key.kdb
   -rw------- 1 adminuser mqm 88 Feb 22 07:50 key.rdb
   ```

   b) If necessary, change group ownership for the key repository files you created so that the mqm group can be given read access. Initially, only the admin user who ran the command has access to the created files.

   ```
   chgrp mqm /var/mqm/qmgrs/qm1/tokenissuer/key.*
   ```

   c) Change the mode of the key repository files to add read permissions for group mqm. For example, the following command adds read/write permissions for the file owner, and read only permission for the group.

   ```
   chmod 640 /var/mqm/qmgrs/qm1/tokenissuer/key.*
   ```

2. Encrypt the key repository password with the **`runqmcred`** command and save the encrypted string to a file.

   a) Create a file to contain the initial key that is used to encrypt the key repository password.

      The file must contain the initial key as a single line of text. The maximum length of the initial key is 256 bytes. If you have already set an initial key for the queue manager by using the **`INITKEY`** queue manager attribute, copy the value of the **`INITKEY`** attribute into the new file. If you have not already set an initial key for the queue manager, create a new, unique encryption key and add it to the initial key file.

      Note: For more information, see INITKEY. If you do not specify the initial key, a default one is used. It is more secure to use your own initial key.

      Note: Grant the minimum necessary permissions on the initial key file to keep the contents of the file secure. The initial key file is only used to encrypt the key repository password. Therefore, only administrators who use the initial key to encrypt passwords need access to the read the initial key file.

   b) If the queue manager initial key is not already set, set the value of the queue manager **`INITKEY`** attribute to the initial key that you created in step "2.a" on page 327. Use the **`ALTER QMGR`** command to set the queue manager initial key. For example:

      ```
      ALTER QMGR INITKEY('myEncrypt10nK3y')
      ```

   c) Issue the **`runqmcred`** command to encrypt the key repository password. Use the **`-sf`** parameter to specify the path to the file that contains the initial key.

      ```
      runqmcred -sf initial.key
      ```

      When prompted, enter the key repository password. The encrypted password is output by the command.

      ```
      5724-H72 (C) Copyright IBM Corp. 1994, 2024.
      Enter password:
      *************
      <QM>!2!b5rb01sMzFzc1ClZeQMryruWFM3HSm8DKyEaZK7qzWY=!TrWdU57DCDXM0Qah99I/Lg==
      ```

      Copy the string on the last line and save it to a file.

3. Use one of the following methods to add the token issuer's public key certificate or symmetric key to the key repository.

   - To add the RSA public key certificate to the key repository, issue the following command:

     ```
     runmqakm -cert -add -db /var/mqm/qmgrs/qm1/tokenissuer/key.kdb -pw MyKeystorePassword
     -label keylabel
              -file keyfile
     ```

   - To add a base64 encoded symmetric key to the key repository, issue the following command:

     ```
     runmqakm -secretkey -add -db /var/mqm/qmgrs/qm1/tokenissuer/key.kdb -pw MyKeystorePassword
     -label keylabel
              -file keyfile -format ascii
     ```

   Where *keylabel* is the label to be attached to the certificate or secret key, and *keyfile* is the name of the file that contains the certificate or the base64 encoded secret key.

4. Add the **`AuthToken`** stanza and the following attributes to the `qm.ini` file:

   - The path to the key repository, specified by using the **`KeyStore`** attribute.
   - The file that contains the password for the key repository, specified by using the **`KeyStorePwdFile`** attribute.
   - The label of the certificate or symmetric key that you added in step "3" on page 327, specified by using the **`CertLabel`** attribute.

   For example:

```
AuthToken:
   KeyStore=/var/mqm/qmgrs/qm1/tokenissuer/key.kdb
   KeyStorePwdFile=/var/mqm/qmgrs/qm1/tokenissuer/key.pw
   CertLabel=rsakey
```

Where `key.kdb` is the name of the key repository that you created in step "1.a" on page 326, and `key.pw` is the file that contains the encrypted password for the key repository that you created in step "2.c" on page 327.

For more information about the **AuthToken** stanza, see AuthToken stanza of the `qm.ini` file.

5. If the queue manager is configured to adopt the user ID that is contained in the token user claim for use in subsequent authorization checks, add the **UserClaim** attribute to the **AuthToken** stanza.

   To determine whether the queue manager is configured to adopt the user ID in the token, issue the following MQSC command:

   ```
   DISPLAY AUTHINFO(authinfo_name) ADOPTCTX
   ```

   Where *authinfo_name* is the value of the queue manager **CONNAUTH** attribute. If the value of the **ADOPTCTX** attribute is YES, the queue manager is configured to adopt the user ID in the token, and the **UserClaim** attribute must be specified in the **AuthToken** stanza.

   Set the value of the **UserClaim** attribute to the name of the token claim that contains the user ID to be adopted. For example, if the token contains the claim `"AppUser": "MyUserName"`, add the following line to the **AuthToken** stanza:

   ```
   UserClaim=AppUser
   ```

6. Refresh the queue manager's security configuration so it picks up the token configuration from the `qm.ini` file. Issue the following command to start the **runmqsc** command:

   ```
   runmqsc qm1
   ```

   then issue the following MQSC command:

   ```
   REFRESH SECURITY TYPE(CONNAUTH)
   ```

What to do next

Work with your developers to help them understand how they can use tokens in applications to authenticate with the queue manager.

Related concepts

Troubleshooting authentication token problems

Related tasks

Using authentication tokens in an application

Related reference

AuthToken stanza of the `qm.ini` file

Linux V 9.4.0 AIX Obtaining an authentication token from your chosen token issuer

Write your application to obtain an authentication token from your chosen token issuer when it connects to an IBM MQ queue manager.

Before you begin

Refer to the information in "Using authentication tokens in an application" on page 329.

Procedure

- How you obtain an authentication token, and the exact contents of the token, varies between different token issuers.

  Write your application to interact with your chosen token issuer to request and obtain the authentication token. The authentication token must conform to the IBM MQ requirements for authentication tokens. For more information about these requirements, see "Requirements for authentication tokens" on page 321.

  If you intend to adopt a user ID that is contained in a token claim as the context for the application, the authentication token must also meet the following requirements:

  - The authentication token must contain a claim that matches the user claim name in the queue manager's token authentication configuration.
  - The value of the user claim must meet the requirements for user IDs in authentication tokens. For more information, see "User IDs in authentication tokens" on page 324.

Results

You have now obtained a correctly formatted JWT which can be presented to IBM MQ for validation.

Related tasks

Configuring a queue manager to accept **AuthTokens**

Related reference

AuthToken stanza of the `qm.ini` file

MQCSP - Security parameters

Linux V 9.4.0 AIX Using authentication tokens in an application

Write your application to supply an authentication token when it connects to an IBM MQ queue manager.

Before you begin

From IBM MQ 9.4.0, applications can supply an authentication token when they connect to a queue manager.

The application must meet the following requirements:

- It must be written in C or Java (using the IBM MQ classes for JMS/ Jakarta Messaging)
- It must connect to the queue manager as an IBM MQ client. That is, the application must connect to the queue manager over a network, instead of using local bindings.
- It must connect to a queue manager that runs on AIX or Linux.

If the application does not meet these requirements, the connection fails and reason code MQRC_FUNCTION_NOT_SUPPORTED (2298) is returned to the application.

The application that supplies the authentication token can run on any platform that supports IBM MQ MQI clients.

Clients that use automatic client reconnection cannot supply an authentication token when they connect. If an application supplies an authentication token, and specifies the `MQCNO_RECONNECT` or `MQCNO_RECONNECT_Q_MGR` option in the MQCNO structure, the connection fails and reason code MQRC_RECONNECT_INCOMPATIBLE (2547) is returned to the application. For more information about automatic client reconnection, see Automatic client reconnection.

If you cannot write the application to supply an authentication token due to these requirements, you can alternatively migrate your application to use authentication tokens by using a client security exit. The client security exit can be written to set the authentication token in the MQCSP structure. For more information about security exits, see Security exits on a client connection.

From IBM MQ 9.4.0, JMS client applications can directly provide a token when connecting (see "Obtaining an authentication token from your chosen token issuer" on page 328). Before IBM MQ 9.4.0, Java applications can indirectly provide a token by way of an exit program. For more information, see Java class MQCSP.

About this task

Note: An authentication token that conforms to the JSON Web Signature (JWS) standard is signed to allow the token's authenticity to be validated, but it is not encrypted. Therefore, it can be read, and possibly reused, by anyone who has access to the token. Configure the connection to the queue manager to ensure that the authentication token is protected by using encryption when it is sent over the network, for example, by using TLS. For more information about the options to protect credentials that are supplied by an application, see "MQCSP password protection" on page 31.

Before modifying applications to connect using a token ensure:

- The queue manager has been configured to accept authentication tokens by following the steps in "Configuring a queue manager to accept authentication tokens using a local keystore" on page 325
- Your application can obtain a valid token as required from your authentication server, see "Obtaining an authentication token from your chosen token issuer" on page 328.

To supply an authentication token when the application connects to an IBM MQ queue manager, include the following process.

Procedure

- To supply an authentication token from a C (MQI) application:

  The application must connect using MQCONNX (rather than MQCONN) and supply an MQCSP structure:

  - The **`AuthenticationType`** field must be set to MQCSP_AUTH_ID_TOKEN.
  - The version of the structure must be set to MQCSP_VERSION_3.
  - The **`TokenPtr`** or **`TokenOffset`** field must reference your authentication token.
  - The **`TokenLength`** field must be set to the length of the authentication token.

  Example C code to connect to a queue manager using MQCSP Version 3 and authentication token:

```
MQCNO cno = {MQCNO_DEFAULT};   /* Connection options  */
MQCSP csp = {MQCSP_DEFAULT};   /* Security parameters */

char  token[MQ_CSP_TOKEN_LENGTH +1] = {0};  /* Authentication token string */

/* Set the connection options */
cno.SecurityParmsPtr = &csp;
cno.Version = MQCNO_VERSION_5;

/* Set the security parameters */
csp.Version = MQCSP_VERSION_3;
csp.AuthenticationType = MQCSP_AUTH_ID_TOKEN;
csp.TokenPtr = token;
csp.TokenLength = (MQLONG) strlen(token);

/* Connect to the queue manager */
MQCONNX(qmName,                 /* Queue manager name  */
        &cno,                   /* Connection options  */
        &hCon,                  /* Connection handle   */
        &compCode,              /* Completion code     */
        &reason);               /* Reason code         */
```

- To supply an authentication token from a Java application:

  Applications using the IBM MQ classes for JMS/Jakarta Messaging can provide a token through any of the `createContext`or `createConnection` methods, which take a username and password.

  To provide an authentication token, the :

  - **`UserID`** must be set to either null or an empty string, that is, without spaces , " "
  - The token is provided as the **`Password`** string.

This applies to all IBM MQ implementations of the `ConnectionFactory` interface.

Either the explicit parameter forms, for example, `createContext`(String **`userID`**, String **`password`**) can be used, or the implicit parameter versions, for example, `createContext`().

In the latter case, the empty **`userID`** and Token **`Password`** must have first been provided as properties on the connection factory.

Example Java code to connect to a queue manager using an authentication token:

```
// Obtain token from authentication provider here:

String myToken = "xxxxxxxxxxxxxxxx";

// Acquire instance of an MQ connection Factory:

JmsFactoryFactory ff = JmsFactoryFactory.getInstance(WMQConstants.WMQ_PROVIDER);

JmsConnectionFactory cf = ff.createConnectionFactory();

// Configure any required CF properties here - e.g. MQ Channel details

// Connect to (and authenticate with) the queue manager:

context = cf.createContext(null, myToken); // NOTE - null userID indicates token being
provided
```

If the connection fails with reason code MQRC_NOT_AUTHORIZED (2035) or MQRC_SECURITY_ERROR (2063), check the queue manager error log for an error message that contains more information about the cause of the failure. For more help with diagnosing problems with authentication tokens, see Troubleshooting authentication token problems.

Results

The application is now connected to the queue manager. It remains connected until it disconnects, even if the token that was used to authenticate expires. If the application disconnects from the queue manager and needs to reconnect, it might need to obtain a new authentication token with a later expiry time before it can reconnect.

Related tasks

Configuring a queue manager to accept **AuthTokens**

Related reference

AuthToken stanza of the `qm.ini` file

MQCSP - Security parameters

Linux V 9.4.0 AIX Creating a key repository for use as a TLS truststore

When creating outgoing TLS connections you should create a simple ‘truststore’ which can validate certificates signed by a common set of certificate authorities (CAs). Example TLS connections are an IBM MQ client channel or an HTTPS connection, as used when configuring some components of IBM MQ.

About this task

Attention: Deciding which certificates and certificate authorities to trust in your environment is an important step with implications for the security of your end to end configuration. This topic is provided to illustrate common steps which allow IBM MQ components to trust the same set of certificates already configured for your operating system; if in doubt, however, you should discuss this process with your security administrator.

Most UNIX and Linux based operating systems have a file system location containing a ‘trusted’ set of CAs. This file system might have been configured with the operating system installation, or customized by your system administrator (for example to include Internal CAs belonging to your organization). Locations for these files vary, but some commonly used values for popular operating systems are:

- AIX: `/var/ssl/cert.pem and/or /var/ssl/certs/*.crt`
- RHEL: `/etc/pki/ca-trust/extracted/pem/tls-ca-bundle.pem`
- Ubuntu: `/etc/ssl/certs/*.pem`

When you create and configure an IBM MQ keystore you can easily add all certificate files in a directory, for example, `/etc/ssl/certs`, to an IBM MQ key database in one command.

Procedure

1. Use the following command to add the certificate files from the `/etc/ssl/certs` directory:

```
runmqakm -cert -add -file /etc/ssl/certs/*.pem -db mykdb.p12 -stashed
```

2. Optional: In some situations, it might be useful to generate a 'default' set of certificates for your trust store.

   The IBM MQ security components provided with the product do provide a set of 'default' CA certificates.

   Note: These certificates might not be frequently updated and/or have relatively short lifetimes.

   If you want to use the preconfigured CA certificates anyway you can generate a truststore using the **`populate`** and **`ibmcloudtrust`** parameters in the **`runmqakm`** command:

```
runmqakm -keydb -create -db mqauto.p12 -genpw -stash -type pkcs12 -populate -ibmcloudtrust
```

Related concepts
Troubleshooting authentication token problems
Related tasks
Using authentication tokens in an application
Related reference
AuthToken stanza of the `qm.ini` file

Working with revoked certificates

Digital certificates can be revoked by Certificate Authorities. You can check the revocation status of certificates using OCSP, or CRLs on LDAP servers, depending on platform.

During the TLS handshake, the communicating partners authenticate each other with digital certificates. Authentication can include a check that the certificate received can still be trusted. Certificate Authorities (CAs) revoke certificates for various reasons, including:

- The owner has moved to a different organization
- The private key is no longer secret

CAs publish revoked personal certificates in a Certificate Revocation List (CRL). CA certificates that have been revoked are published in an Authority Revocation List (ARL).

ALW On AIX, Linux, and Windows platforms, IBM MQ SSL support checks for revoked certificates using OCSP (Online Certificate Status Protocol) or using CRLs and ARLs on LDAP (Lightweight Directory Access Protocol) servers. OCSP is the preferred method.

IBM MQ classes for Java and IBM MQ classes for JMS cannot use the OCSP information in a client channel definition table file. However, you can configure OCSP as described in Using Online Certificate Protocol.

IBM i On IBM i, IBM MQ SSL support checks for revoked certificates using CRLs and ARLs on LDAP servers only.

z/OS On z/OS, IBM MQ SSL support checks for revoked certificates using CRLs and ARLs on LDAP servers only.

For more information about Certificate Authorities, see "Digital certificates" on page 13.

OCSP/CRL checking

Online Certificate Status Protocol (OCSP)/Certificate Revocation List (CRL) checking is performed against remote incoming certificates. The process checks the whole chain involved from the personal certificate of the remote system right through to its root certificate.

Using openSSL to verify OCSP validation

If your enterprise uses openSSL to validate OCSP, and then you attempt to use a IBM Global Security Kit (GSKit) TLS connection, you receive an UNKNOWN status warning.

This is because all certificates in the chain, apart from the root, are checked by GSKit for revocation status. GSKit operation is in accordance with RFC 5280 and this is described in the GSKit Trust Policy. The GSKit algorithm tries all available sources for revocation information, as described in RFC 5280 and the GSKit Trust Policy.

How does the OCSP/CRL checking work in IBM MQ?

IBM MQ supports two mechanisms for controlling behavior when checking certificates against named OCSP or CRL endpoints, either in the certificate extension or, as defined in the AUTHINFO objects:

- The **`OCSPCheckExtensions`**, **`CDPCheckExtensions`**, and **`OCSPAuthentication`** attributes of the SSL stanza of the qm.ini file, and
- Using the SSLCRLNL parameter of the queue manager and the AUTHINFO OCSP and CRLLDAP configurations. See ALTER AUTHINFO and ALTER QMGR for more information.

Attention:

The ALTER AUTHINFO command with **`AUTHTYPE(OCSP)`** does not apply for use on IBM i or z/OS queue managers. However, it can be specified on those platforms to be copied to the client channel definition table (CCDT) for client use.

The **`OCSPCheckExtensions`** and **`CDPCheckExtensions`** SSL stanza attributes control whether IBM MQ will verify a certificate against the OCSP or CRL server detailed inside the AIA extension of the certificate.

If not enabled, the OCSP or CRL server in the certificate extension is not contacted.

If OCSP or CRL servers are detailed through AUTHINFO objects, and referenced using the SSLCRLNL **`QMGR`** attribute then, during certificate revocation processing, IBM MQ attempts to contact these servers.

Important: Only one OCSP AUTHINFO object can be defined in the SSLCRLNL namelist.

If:

`OCSPCheckExtensions`=NO and **`CDPCheckExtensions`**=*NO* are set, and
No OCSP or CRL servers are defined in AUTHINFO objects

no certificate revocation checking is performed.

When verifying a certificate for its revocation status, IBM MQ contacts the OCSP or CRL servers named in the following order, if enabled:

1. The OCSP server detailed in an **`AUTHTYPE(OCSP)`** object, and referenced in the SSLCRLNL **`QMGR`** attribute.
2. OCSP servers detailed in the AIA extension of the certificates, if **`OCSPCheckExtensions`**=*YES*.
3. CRL servers detailed in the **`CRLDistributionPoints`** extension of the certificates, if **`CDPCheckExtensions`** =*YES*.
4. Any CRL servers detailed in **`AUTHINFO(CRLLDAP)`** objects and referenced in the SSLCRLNL **`QMGR`** attribute.

While verifying a certificate, if a step results in the OCSP server or CRL server returning a definitive `REVOKED` or `VALID` response to a query for the certificate, no further checks are performed and the status of the certificate as presented is used to determine whether to trust it or not.

If an OCSP server or CRL server returns a result of `UNKNOWN`, processing continues until an OCSP or CRL server returns a definitive result, or all options are exhausted.

The behavior of whether a certificate is considered revoked, if its status cannot be determined, is different for OCSP and CRL servers:

- For CRL servers, if no CRL can be obtained, the certificate is considered `NOT_REVOKED`
- For OCSP servers, if no revocation status can be obtained from a named OCSP server then the behavior is controlled through the **`OCSPAuthentication`** attribute in the SSL Stanza of the qm.ini file.

  You can configure this attribute to either, block a connection, allow a connection, or allow a connection with a warning message.

You can use the **`SSLHTTPProxyName`**=*string* attribute in the SSL stanza of the qm.ini and mqclient.ini files for the OCSP checks if needed. The string is either the host name, or network address of the HTTP Proxy server that is to be used by GSKit for OCSP checks.

You can set the **`OCSPTimeout`** value in the SSL stanza of the `qm.ini` or `mqclient.ini` files that sets the number of seconds to wait for an OCSP responder when performing a revocation check.

ALW Revoked certificates and OCSP

IBM MQ determines which Online Certificate Status Protocol (OCSP) responder to use, and handles the response received. You might have to take steps to make the OCSP responder accessible.

Note: This information applies only to IBM MQ on AIX, Linux, and Windows systems.

To check the revocation status of a digital certificate using OCSP, IBM MQ can use two methods to determines which OCSP responder to contact:

- By using the AuthorityInfoAccess (AIA) certificate extension in the certificate to be checked.
- By using a URL specified in an authentication information object or specified by a client application.

A URL specified in an authentication information object or by a client application takes priority over a URL in an AIA certificate extension.

If the URL of the OCSP responder lies behind a firewall, reconfigure the firewall so the OCSP responder can be accessed or set up an OCSP proxy server. Specify the name of the proxy server by using the SSLHTTPProxyName variable in the SSL stanza. On client systems, you can also specify the name of the proxy server by using the environment variable MQSSLPROXY. For more details, see the related information.

If you are not concerned whether TLS certificates are revoked, perhaps because you are running in a test environment, you can set OCSPCheckExtensions to NO in the SSL stanza. If you set this variable, any AIA certificate extension is ignored. This solution is unlikely to be acceptable in a production environment, where you probably do not want to allow access from users presenting revoked certificates.

The call to access the OCSP responder can result in one of the following three outcomes:

Good
The certificate is valid.

Revoked
The certificate is revoked.

Unknown
This outcome can arise for one of three reasons:

- IBM MQ cannot access the OCSP responder.
- The OCSP responder has sent a response, but IBM MQ cannot verify the digital signature of the response.
- The OCSP responder has sent a response that indicates that it has no revocation data for the certificate.

If IBM MQ receives an OCSP outcome of `Unknown`, its behavior depends on the setting of the OCSPAuthentication attribute. For queue managers, this attribute is held in one of the following locations:

- Linux AIX In the SSL stanza of the `qm.ini` file on AIX and Linux.
- Windows In the Windows registry.

This attribute can be set using the IBM MQ Explorer. For clients, the attribute is held in the SSL stanza of the client configuration file.

If an outcome of `Unknown` is received and OCSPAuthentication is set to REQUIRED (the default value), IBM MQ rejects the connection and issues an error message of type AMQ9716. If queue manager SSL event messages are enabled, an SSL event message of type MQRC_CHANNEL_SSL_ERROR with ReasonQualifier set to MQRQ_SSL_HANDSHAKE_ERROR is generated.

If an outcome of `Unknown` is received and OCSPAuthentication is set to OPTIONAL, IBM MQ allows the SSL channel to start and no warnings or SSL event messages are generated.

If an outcome of `Unknown` is received and OCSPAuthentication is set to WARN, the SSL channel starts but IBM MQ issues a warning message of type AMQ9717 in the error log. If queue manager SSL event messages are enabled, an SSL event message of type MQRC_CHANNEL_SSL_WARNING with ReasonQualifier set to MQRQ_SSL_UNKNOWN_REVOCATION is generated.

Digital signing of OCSP responses

An OCSP responder can sign its responses in one of three ways. Your responder will inform you which method is used.

- The OCSP response can be digitally signed using the same CA certificate that issued the certificate that you are checking. In this case, you do not need to set up any additional certificate; the steps you have already taken to establish TLS connectivity are sufficient to verify the OCSP response.
- The OCSP response can be digitally signed using another certificate signed by the same certificate authority (CA) that issued the certificate you are checking. The signing certificate is sent together with the OCSP response in this case. The certificate flowed from the OCSP responder must have an Extended Key Usage Extension set to `id-kp-OCSPSigning` so that it can be trusted for this purpose. Because the OCSP response is sent with the certificate which signed it (and that certificate is signed by a CA that is already trusted for TLS connectivity), no additional certificate setup is required.
- The OCSP response can be digitally signed using another certificate that is not directly related to the certificate you are checking. In this case, the OCSP response is signed by a certificate issued by the OCSP responder itself. You must add a copy of the OCSP responder certificate to the key database of the client or queue manager that performs the OCSP checking. see "Adding a CA certificate, or the public part of a trusted certificate, into a key repository on AIX, Linux, and Windows" on page 537. When a CA certificate is added, by default it is added as a trusted root, which is the required setting in this context. If this certificate is not added, IBM MQ cannot verify the digital signature on the OCSP response and the OCSP check results in an Unknown outcome, which might cause IBM MQ to close the channel, depending on the value of OCSPAuthentication.

Online Certificate Status Protocol (OCSP) in Java and JMS client applications

Due to a limitation of the Java API, IBM MQ can use Online Certificate Status Protocol (OCSP) certificate revocation checking for TLS secure sockets only when OCSP is enabled for the entire Java virtual machine (JVM) process. There are two ways to enable OCSP for all secure sockets in the JVM:

- Edit the JRE java.security file to include the OCSP configuration settings that are shown in Table 1 and restart the application.
- Use the java.security.Security.setProperty() API, subject to any Java Security Manager policy in effect.

As a minimum, you must specify one of the ocsp.enable and ocsp.responderURL values.

Property Name	Description
ocsp.enable	This property's value is either `true` or `false`. If `true`, OCSP checking is enabled when doing certificate revocation checking; if `false` or not set, OCSP checking is disabled.
ocsp.responderURL	This property's value is a URL that identifies the location of the OCSP responder. Here is an example; `ocsp.responderURL=http://ocsp.example.net:80`. By default, the location of the OCSP responder is determined implicitly from the certificate that is being validated. The property is used when the Authority Information Access extension (defined in RFC 3280) is absent from the certificate or when it requires overriding.
ocsp.responderCertSubjectName	This property's value is the subject name of the OCSP responder's certificate. Here is an example; `ocsp.responderCertSubjectName="CN=OCSP Responder, O=XYZ Corp"`. By default, the certificate of the OCSP responder is that of the issuer of the certificate that is being validated. This property identifies the certificate of the OCSP responder when the default does not apply. Its value is a string distinguished name (defined in RFC 2253) which identifies a certificate in the set of certificates that are supplied during cert path validation. In cases where the subject name alone is not sufficient to uniquely identify the certificate, then both the ocsp.responderCertIssuerName and ocsp.responderCertSerialNumber properties must be used instead. When this property is set, then the properties ocsp.responderCertIssuerName and ocsp.responderCertSerialNumber are ignored.
ocsp.responderCertIssuerName	This property's value is the issuer name of the OCSP responder's certificate. Here is an example; `ocsp.responderCertIssuerName="CN=Enterprise CA, O=XYZ Corp"`. By default, the certificate of the OCSP responder is that of the issuer of the certificate that is being validated. This property identifies the certificate of the OCSP responder when the default does not apply. Its value is a string distinguished name (defined in RFC 2253) which identifies a certificate in the set of certificates that are supplied during cert path validation. When this property is set then the ocsp.responderCertSerialNumber property must also be set. This property is ignored when the ocsp.responderCertSubjectName property is set.
ocsp.responderCertSerialNumber	This property's value is the serial number of the OCSP responder's certificate. Here is an example; `ocsp.responderCertSerialNumber=2A:FF:00`. By default, the certificate of the OCSP responder is that of the issuer of the certificate that is being validated. This property identifies the certificate of the OCSP responder when the default does not apply. This value is a string of hexadecimal digits (colon or space separators might be present) which identifies a certificate in the set of certificates that are supplied during cert path validation. When this property is set then the ocsp.responderCertIssuerName property must also be set. This property is ignored when the ocsp.responderCertSubjectName property is set.

Before you enable OCSP in this way, there are a number of considerations:

- Setting the OCSP configuration affects all secure sockets in the JVM process. In some cases this configuration might have undesirable side-effects when the JVM is shared with other application code that uses TLS secure sockets. Ensure that the chosen OCSP configuration is suitable for all of the applications that are running in the same JVM.

- Applying maintenance to your JRE might overwrite the java.security file. Take care when you apply Java interim fixes and product maintenance to avoid overwriting the java.security file. It might be necessary to reapply your java.security changes after you apply maintenance. For this reason, you might consider setting the OCSP configuration by using the java.security.Security.setProperty() API instead.
- Enabling OCSP checking has an effect only if revocation checking is also enabled. Revocation checking is enabled by the `PKIXParameters.setRevocationEnabled()` method.
- If you are using the AMS Java Interceptor described in Enabling OCSP checking in native interceptors, take care to avoid using a java.security OCSP configuration that conflicts with the AMS OCSP configuration in the keystore configuration file.

Working with Certificate Revocation Lists and Authority Revocation Lists

IBM MQ support for CRLs and ARLs varies by platform.

CRL and ARL support on each platform is as follows:

- Multi On Multiplatforms, the CRL and ARL support complies with PKIX X.509 V2 CRL profile recommendations.
- z/OS On z/OS, System SSL supports CRLs and ARLs stored in LDAP servers by the Tivoli Public Key Infrastructure product.

IBM MQ maintains a cache of CRLs and ARLs that have been accessed in the preceding 12 hours.

When a queue manager or IBM MQ MQI client receives a certificate, it checks the CRL to confirm that the certificate is still valid. IBM MQ first checks in the cache, if there is a cache. If the CRL is not in the cache, IBM MQ interrogates the LDAP CRL server locations in the order they occur in the namelist of authentication information objects specified by the *SSLCRLNL* attribute, until IBM MQ finds an available CRL. If the namelist is not specified, or is specified with a blank value, CRLs are not checked.

Setting up LDAP servers

Configure the LDAP Directory Information Tree structure to reflect the hierarchy of Distinguished Names of CAs. Do this using LDAP Data Interchange Format files.

Configure the LDAP Directory Information Tree (DIT) structure to use the hierarchy corresponding to the Distinguished Names of the CAs that issue the certificates and CRLs. You can set up the DIT structure with a file that uses the LDAP Data Interchange Format (LDIF). You can also use LDIF files to update a directory.

LDIF files are ASCII text files that contain the information required to define objects within an LDAP directory. LDIF files contain one or more entries, each of which comprises a Distinguished Name, at least one object class definition and, optionally, multiple attribute definitions.

The `certificateRevocationList;binary` attribute contains a list, in binary form, of revoked user certificates. The `authorityRevocationList;binary` attribute contains a binary list of CA certificates that have been revoked. For use with IBM MQ TLS, the binary data for these attributes must conform to DER (Definite Encoding Rules) format. For more information about LDIF files, refer to the documentation provided with your LDAP server.

Figure 20 on page 338 shows a sample LDIF file that you might create as input to your LDAP server to load the CRLs and ARLs issued by CA1, which is an imaginary Certificate Authority with the Distinguished Name "CN=CA1, OU=Test, O=IBM, C=GB", set up by the Test organization within IBM.

```
dn: o=IBM, c=GB
o: IBM
objectclass: top
objectclass: organization

dn: ou=Test, o=IBM, c=GB
ou: Test
objectclass: organizationalUnit

dn: cn=CA1, ou=Test, o=IBM, c=GB
cn: CA1
objectclass: cRLDistributionPoint
objectclass: certificateAuthority
authorityRevocationList;binary:: (DER format data)
certificateRevocationList;binary:: (DER format data)
caCertificate;binary:: (DER format data)
```

Figure 20. Sample LDIF file for a Certificate Authority. This might vary from implementation to implementation.

Figure 21 on page 338 shows the DIT structure that your LDAP server creates when you load the sample LDIF file shown in Figure 20 on page 338 together with a similar file for CA2, an imaginary Certificate Authority set up by the PKI organization, also within IBM.

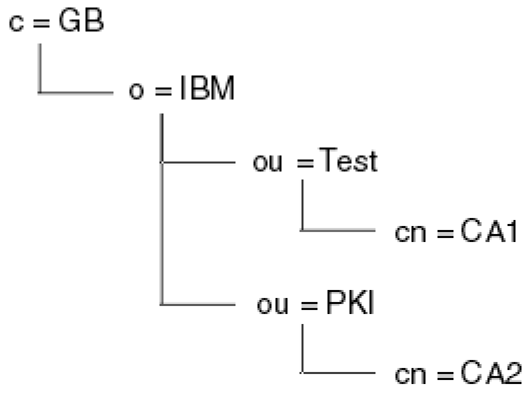


Figure 21. Example of an LDAP Directory Information Tree structure

IBM MQ checks both CRLs and ARLs.

Note: Ensure that the access control list for your LDAP server allows authorized users to read, search, and compare the entries that hold the CRLs and ARLs. IBM MQ accesses the LDAP server using the LDAPUSER and LDAPPWD properties of the AUTHINFO object.

Configuring and updating LDAP servers
Use this procedure to configure or update your LDAP server.

1. Obtain the CRLs and ARLs in DER format from your Certification Authority, or Authorities.
2. Using a text editor or the tool provided with your LDAP server, create one or more LDIF files that contain the Distinguished Name of the CA and the required object class definitions. Copy the DER format data into the LDIF file as the values of either the `certificateRevocationList;binary` attribute for CRLs, the `authorityRevocationList;binary` attribute for ARLs , or both.
3. Start your LDAP server.
4. Add the entries from the LDIF file or files you created at step "2" on page 338.

After you have configured your LDAP CRL server, check that it is set up correctly. First, try using a certificate that is not revoked on the channel, and check that the channel starts correctly. Then use a certificate that is revoked, and check that the channel fails to start.

Obtain updated CRLs from the Certification Authorities frequently. Consider doing this on your LDAP servers every 12 hours.

Accessing CRLs and ARLs with a queue manager

A queue manager is associated with one or more authentication information objects, which hold the address of an LDAP CRL server. IBM i IBM MQ on IBM i behaves differently from other platforms.

Note that in this section, information about Certificate Revocation Lists (CRLs) also applies to Authority Revocation Lists (ARLs).

You tell the queue manager how to access CRLs by supplying the queue manager with authentication information objects, each of which holds the address of an LDAP CRL server. The authentication information objects are held in a namelist, which is specified in the *SSLCRLNL* queue manager attribute.

In the following example, MQSC is used to specify the parameters:

1. Define authentication information objects using the DEFINE AUTHINFO MQSC command, with the AUTHTYPE parameter set to CRLLDAP. IBM i On IBM i, you can also use the CRTMQMAUTI CL command.

   The value CRLLDAP for the AUTHTYPE parameter indicates that CRLs are accessed on LDAP servers. Each authentication information object with type CRLLDAP that you create holds the address of an LDAP server. When you have more than one authentication information object, the LDAP servers to which they point must contain identical information. This provides continuity of service if one or more LDAP servers fail.

   z/OS Additionally, on z/OS only, all LDAP servers must be accessed using the same user ID and password. The user ID and password used are those specified in the first AUTHINFO object in the namelist.

   On all platforms, the user ID and password are sent to the LDAP server unencrypted.
2. Using the DEFINE NAMELIST MQSC command, define a namelist for the names of your authentication information objects. z/OS On z/OS, ensure that the NLTYPE namelist attribute is set to AUTHINFO.
3. Using the ALTER QMGR MQSC command, supply the namelist to the queue manager. For example:

   ```
   ALTER QMGR SSLCRLNL(sslcrlnlname)
   ```

   where `sslcrlnlname` is your namelist of authentication information objects.

   This command sets a queue manager attribute called *SSLCRLNL*. The queue manager's initial value for this attribute is blank.

IBM i On IBM i, you can specify authentication information objects, but the queue manager uses neither authentication information objects nor a namelist of authentication information objects. Only IBM MQ clients that use a client connection table generated by an IBM i queue manager use the authentication information specified for that IBM i queue manager. The *SSLCRLNL* queue manager attribute on IBM i determines what authentication information such clients use. See "Accessing CRLs and ARLs on IBM i" on page 339 for information about telling an IBM i queue manager how to access CRLs.

You can add up to 10 connections to alternative LDAP servers to the namelist, to ensure continuity of service if one or more LDAP servers fail. Note that the LDAP servers must contain identical information.

IBM i *Accessing CRLs and ARLs on IBM i*

Use this procedure to access CRLs or ARLs on IBM i.

Note that in this section, information about Certificate Revocation Lists (CRLs) also applies to Authority Revocation Lists (ARLs).

Follow these steps to set up a CRL location for a specific certificate on IBM i:

1. Access the DCM interface, as described in "Accessing DCM" on page 270.
2. In the **Manage CRL locations** task category in the navigation panel, click **Add CRL location**. The Manage CRL Locations page is displayed in the task frame.
3. In the **CRL Location Name** field, type a CRL location name, for example `LDAP Server #1`
4. In the **LDAP Server** field, type the LDAP server name.
5. In the **Use Secure Sockets Layer (SSL)** field, select **Yes** if you want to connect to the LDAP server using TLS. Otherwise, select **No**.
6. In the **Port Number** field, type a port number for the LDAP server, for example 389.
7. If your LDAP server does not allow anonymous users to query the directory, type a login distinguished name for the server in the **login distinguished name** field.
8. Click **OK**. DCM informs you that it has created the CRL location.
9. In the navigation panel, click **Select a Certificate Store**. The Select a Certificate Store page is displayed in the task frame.
10. Select the **Other System Certificate Store** check box and click **Continue**. The Certificate Store and Password page is displayed.
11. In the **Certificate store path and filename** field, type the IFS path and file name you set when "Creating a certificate store on IBM i" on page 273.
12. Type a password in the **Certificate Store Password** field. Click **Continue**. The Current Certificate Store page is displayed in the task frame.
13. In the **Manage Certificates** task category in the navigation panel, click **Update CRL location assignment**. The CRL Location Assignment page is displayed in the task frame.
14. Select the radio button for the CA certificate to which you want to assign the CRL location. Click **Update CRL Location Assignment**. The Update CRL Location Assignment page is displayed in the task frame.
15. Select the radio button for the CRL location which you want to assign to the certificate. Click **Update Assignment**. DCM informs you that it has updated the assignment.

Note that DCM allows you to assign a different LDAP server by Certificate Authority.

Accessing CRLs and ARLs using IBM MQ Explorer
You can use IBM MQ Explorer to tell a queue manager how to access CRLs.

Note that in this section, information about Certificate Revocation Lists (CRLs) also applies to Authority Revocation Lists (ARLs).

Use the following procedure to set up an LDAP connection to a CRL:

1. Ensure that you have started your queue manager.
2. Right-click the **Authentication Information** folder and click **New -> Authentication Information**. In the property sheet that opens:
   a. On the first page **Create Authentication Information**, enter a name for the CRL(LDAP) object.
   b. On the **General** page of **Change Properties**, select the connection type. Optionally you can enter a description.
   c. Select the **CRL(LDAP)** page of **Change Properties**.
   d. Enter the LDAP server name as either the network name or the IP address.
   e. If the server requires login details, provide a user ID and if necessary a password.
   f. Click **OK**.
3. Right-click the Namelists folder and click **New -> Namelist**. In the property sheet that opens:
   a. Type a name for the namelist.
   b. Add the name of the CRL(LDAP) object (from step "2.a" on page 340) to the list.
   c. Click **OK**.

4. Right-click the queue manager, select **Properties**, and select the **SSL** page:
   a. Select the **Check certificates received by this queue manager against Certification Revocation Lists** check box.
   b. Type the name of the namelist (from step "3.a" on page 340) in the **CRL Namelist** field.

Accessing CRLs and ARLs with an IBM MQ MQI client

You have three options for specifying the LDAP servers that hold CRLs for checking by an IBM MQ MQI client.

Note that in this section, information about Certificate Revocation Lists (CRLs) also applies to Authority Revocation Lists (ARLs).

The three ways of specifying the LDAP servers are as follows:

- Using a channel definition table
- Using the SSL configuration options structure, MQSCO, on an MQCONNX call
- Using the Active Directory (on Windows systems with Active Directory support)

For more details, refer to the related information.

You can include up to 10 connections to alternative LDAP servers to ensure continuity of service if one or more LDAP servers fail. Note that the LDAP servers must contain identical information.

You cannot access LDAP CRLs from an IBM MQ MQI client channel running on Linux (zSeries platform).

Location of an OCSP responder, and of LDAP servers that hold CRLs
On an IBM MQ MQI client system, you can specify the location of an OCSP responder, and of Lightweight Directory Access Protocol (LDAP) servers that hold certificate revocation lists (CRLs).

You can specify these locations in three ways, described here in order of decreasing precedence.

IBM i For IBM i, see Accessing CRLs and ARLs on IBM i.

When an IBM MQ MQI client application issues an MQCONNX call

You can specify an OCSP responder or an LDAP server holding CRLs on an **MQCONNX** call.

On an **MQCONNX** call, the connect options structure, MQCNO, can reference an SSL configuration options structure, MQSCO. In turn, the MQSCO structure can reference one or more authentication information record structures, MQAIR. Each MQAIR structure contains all the information an IBM MQ MQI client requires to access an OCSP responder or an LDAP server holding CRLs. For example, one of the fields in an MQAIR structure is the URL at which a responder can be contacted. For more information about the MQAIR structure, see MQAIR - Authentication information record.

Using a client channel definition table (ccdt) to access an OCSP responder or LDAP servers

So that an IBM MQ MQI client can access an OCSP responder or LDAP servers that hold CRLs, include the attributes of one or more authentication information objects in a client channel definition table.

On a server queue manager, you can define one or more authentication information objects. The attributes of an authentication object contain all the information that is required to access an OCSP responder (on platforms where OCSP is supported) or an LDAP server that holds CRLs. One of the attributes specifies the OCSP responder URL, another specifies the host address, or IP address of a system on which an LDAP server runs.

z/OS IBM i An authentication information object with AUTHTYPE(OCSP) does not apply for use on IBM i or z/OS queue managers, but it can be specified on those platforms to be copied to the client channel definition table (CCDT) for client use.

To enable an IBM MQ MQI client to access an OCSP responder or LDAP servers that hold CRLs, the attributes of one or more authentication information objects can be included in a client channel definition table. You can include such attributes in one of the following ways:

Multi

On server platforms AIX, Linux, IBM i, and Windows

You can define a namelist that contains the names of one or more authentication information objects. You can then set the queue manager attribute, **SSLCRLNL**, to the name of this namelist.

If you are using CRLs, more than one LDAP server can be configured to provide higher availability. The intention is that each LDAP server holds the same CRLs. If one LDAP server is unavailable when it is required, an IBM MQ MQI client can attempt to access another.

The attributes of the authentication information objects identified by the namelist are referred to collectively here as the *certificate revocation location*. When you set the queue manager attribute, **SSLCRLNL**, to the name of the namelist, the certificate revocation location is copied into the client channel definition table associated with the queue manager. If the CCDT can be accessed from a client system as a shared file, or if the CCDT is then copied to a client system, the IBM MQ MQI client on that system can use the certificate revocation location in the CCDT to access an OCSP responder or LDAP servers that hold CRLs.

If the certificate revocation location of the queue manager is changed later, the change is reflected in the CCDT associated with the queue manager. If the queue manager attribute, **SSLCRLNL**, is set to blank, the certificate revocation location is removed from the CCDT. These changes are not reflected in any copy of the table on a client system.

If you require the certificate revocation location at the client and server ends of an MQI channel to be different, and the server queue manager is the one that is used to create the certificate revocation location, you can do it as follows:

1. On the server queue manager, create the certificate revocation location for use on the client system.
2. Copy the CCDT containing the certificate revocation location to the client system.
3. On the server queue manager, change the certificate revocation location to what is required at the server end of the MQI channel.
4. On the client machine, you can use the **runmqsc** command with the **-n** parameter.

Multi

On client platforms AIX, Linux, IBM i, and Windows

You can build a CCDT on the client machine by using the runmqsc command with the **-n** parameter and **DEFINE AUTHINFO** objects in the CCDT file. The order that the objects are defined in is the order in which they are used in the file. Any name that you might use in a **DEFINE AUTHINFO** object is not retained in the file. Only positional numbers are used when you **DISPLAY** the **AUTHINFO** objects in a CCDT file.

Note: If you specify the **-n** parameter, you must not specify any other parameter.

Using Active Directory on Windows

Windows

On Windows systems, you can use the **setmqcrl** control command to publish the current CRL information in Active Directory.

Command **setmqcrl** does not publish OCSP information.

For information about this command and its syntax, see setmqcrl.

Accessing CRLs and ARLs with IBM MQ classes for Java and IBM MQ classes for JMS

IBM MQ classes for Java and IBM MQ classes for JMS access CRLs differently from other platforms.

For information about working with CRLs and ARLs with IBM MQ classes for Java, see Using certificate revocation lists

For information about working with CRLs and ARLs with IBM MQ classes for JMS, see SSLCERTSTORES object property

Manipulating authentication information objects

You can manipulate authentication information objects using MQSC or PCF commands, or the IBM MQ Explorer.

The following MQSC commands act on authentication information objects:

- DEFINE AUTHINFO
- ALTER AUTHINFO
- DELETE AUTHINFO
- DISPLAY AUTHINFO

For a complete description of these commands, see MQSC commands.

The following Programmable Command Format (PCF) commands act on authentication information objects:

- Create Authentication Information
- Copy Authentication Information
- Change Authentication Information
- Delete Authentication Information
- Inquire Authentication Information
- Inquire Authentication Information Names

For a complete description of these commands, see Definitions of the Programmable Command Formats.

On platforms where it is available, you can also use the IBM MQ Explorer.

Linux AIX Using the Pluggable Authentication Method (PAM)

You can use PAM only on AIX and Linux platforms. A typical AIX or Linux system has PAM modules that implement the traditional authentication mechanism; however, there might be more. As well as the basic task of validating passwords, PAM modules can also be invoked to carry out additional rules.

Configuration files define which authentication method is to be used for each application . Example applications include the standard terminal login, ftp, and telnet.

The advantage of PAM is that the application does not need to know, or care about, how the user ID is actually being authenticated. As long as the application can provide a correct form of authentication data to PAM, the mechanism behind it is transparent.

The form of authentication data depends upon the system being used. For example, IBM MQ obtains a password through parameters, such as the MQCSP structure used in the `MQCONNX` API call.

Important: You cannot set the **`AUTHENMD`** attribute until you install IBM MQ 8.0.0 Fix Pack 3, and then restart the queue manager, using a **`-e CMDLEVEL=`***level* of *802* (on the strmqm command) to set the command level you require.

Configuring your system to use PAM

The service name used by IBM MQ, when invoking PAM, is *ibmmq*.

Note that an IBM MQ installation attempts to maintain a default PAM configuration, that permits connections from operating system users, based on known defaults for the different operating systems.

However, your system administrator must verify that rules defined in the `/etc/pam.conf`, or `/etc/pam.d/ibmmq`, files are still appropriate.

Authorizing access to objects

This section contains information about using the object authority manager and channel exit programs to control access to objects.

ALW On AIX, Linux, and Windows systems. you control access to objects by using the object authority manager (OAM). This collection of topics contains information about using the command interface to the OAM.

This section also contains a checklist you can use to determine what tasks to perform to apply security to your system on all platforms, and considerations for granting users the authority to administer IBM MQ and to work with IBM MQ objects.

If the supplied security mechanisms do not meet your needs, you can develop your own channel exit programs.

Determining which user is used for authorization

Authorities to access resources are granted to groups that the user is a member of or, in certain modes, directly to the user associated with the connection. During the connection process, and in particular for remote (client) connections, this identity could be changed by the queue manager's configuration. This page lists the different features of IBM MQ and their configuration options that could impact a connecting application's identity and the order of precedence in which these features take effect.

Features that can modify which user is adopted

The different features that can set which user should be authorized are as follows:

Application asserted user

When a remote connection is started by IBM MQ, the operating system user that the process is running as is sent to the receiving queue manager. This user is sent to ensure that if no further configuration exists that modifies the user, there is a user that can be used for authorization checking.

It is not recommended to use this user as the basis for authorization as it allows connections to assert their identity without any server-side validation. This might even include the administrative user ('mqm').

Channel MCAUSER setting

Applications connecting through network bindings do so by using an IBM MQ channel definition. Channel definitions support the **MCAUSER** attribute, which can be used to specify a different user to be used for authorization instead of the user asserted by the connecting applications.

Connection authentication ADOPTCTX

Applications can specify a user and password to be sent to a queue manager for authentication purposes. These credentials are authenticated using the configuration that is specified for the Connection Authentication feature. The **ADOPTCTX** option for Connection Authentication controls whether a user should be used for authorization after it has been successfully validated. If set to YES, then the user that is supplied for authentication is adopted for authorization checks.

V 9.4.0 From IBM MQ 9.3.4, a token can be supplied for authentication, if **ADOPTCTX** is set to YES, then a user is adopted from the claims that the token contains.

Channel authentication record MCAUSER

During connection processing the queue manager will attempt to find a channel authentication record that matches the connection. If a channel authentication record is matched, and its **USERSRC** attribute value is set to MAP, then IBM MQ changes the user used for authorizations to the value of the **MCAUSER** attribute.

Security exits

Security exits are custom functions that can be written and called during the IBM MQ security processing. When the function is called it is supplied with a copy of the MQCD structure that includes several fields relating to the connections user that will be used for authorization checks. Security exits can modify these fields to change the user that will be authorized.

Order of precedence

The following table shows the order of precedence for each security feature described in "Features that can modify which user is adopted" on page 344 when IBM MQ is selecting a user to authorize. The order is from lowest to highest, that is, a security feature setting a user at the first row is overridden by any of the other rows.

Table 71. Order of precedence for security features

Order	Feature
1 (lowest)	Application Asserted ID
2	Channel definition **MCAUSER** attribute
3	Connection authentication with **ADOPTCTX(YES)**
4	Channel authentication records with **USERSRC(MAP)**
5 (highest)	Security exit

Implications of early adopt

Connection authentication and channel authentication records provide a configuration option that controls when connection authentication user adoption is performed. This setting is referred to as early adopt. If early adopt is enabled, connection authentication identity adoption happens before channel authentication records are processed (meaning the channel authentication records override any **CONNAUTH** adoption).

If disabled, the order is reversed – that is, channel authentication records are processed before **CONNAUTH** adoption. In this situation, connection authentication adoption has a higher effective priority that channel authentication records.

The default setting for early adopt is `enabled`.

ALW Controlling access to objects by using the OAM on AIX, Linux, and Windows

The object authority manager (OAM) provides a command interface for granting and revoking authority to IBM MQ objects.

You must be suitably authorized to use these commands, as described in "Authority to administer IBM MQ on AIX, Linux, and Windows" on page 392. User IDs that are authorized to administer IBM MQ have *super user* authority to the queue manager, which means that you do not have to grant them further permission to issue any MQI requests or commands.

Linux AIX OAM user-based permissions on AIX and Linux

On UNIX and Linux systems, the object authority manager (OAM) can use user-based authorization as well as group-based authorization.

Before IBM MQ 8.0, access control lists (ACLs) on UNIX and Linux are based on groups only. From IBM MQ 8.0, ACLs are based on both user IDs and groups, and you can use either the user-based model or the group-based model for authorization by setting the **SecurityPolicy** attribute to the appropriate value as described in Service stanza of the qm.ini file.

Changes in behavior for IBM MQ 8.0 and later

From IBM MQ 8.0, when running with the user-based policy, some commands return different information from earlier versions of the product:

- The **dmpmqaut** and **dmpmqcfg** commands show user-based records, as do the PCF equivalent operations.
- The OAM plug-in for IBM MQ Explorer shows user-based records and allows user-based modifications.
- The OAM **Inquire** function returns results that show that it is user-capable.

Using the **-p** attribute on the **setmqaut** command does not grant access to all users in the same primary group, when user-based authorizations are enabled in the `qm.ini` file as described in Service stanza of the qm.ini file.

If you start to employ user-based authorization and have many users, there will probably be more records that are stored on the AUTH queue than with the group-based model, and the authorization process might take a little longer than previously as there are more records to verify. This increase is not expected to be significant. If required, you can use a mixture of user and group permissions.

Migration considerations

If you change the model from group to user for an existing queue manager, there is no immediate effect. The authorizations that have already been made continue to apply. Any user that connects to the queue manager receives the same privileges as before: the combination of all the groups to which their ID belongs. When new **setmqaut** commands are issued for user IDs, they take immediate effect.

If you create a new queue manager with the user policy, this queue manager has permissions only for the user who creates it (which is normally, but not necessarily, the `mqm` user ID). There are also permissions that are automatically granted to the `mqm` group. However, if you do not have `mqm` as the primary group, then the `mqm` group is not included in the initial set of authorizations.

If you move from a user to group policy, the user-based authorizations are not automatically deleted. However, they are no longer used during the permissions check. Before reverting the policy, save the current configuration, change the policy, restart the queue manager, and then replay the script. Because it is now a group-based queue manager, the effect is that user ID rules are stored based on the primary group.

Related concepts

Object authority manager (OAM)

"Principals and groups on AIX, Linux, and Windows" on page 397
Principals can belong to groups. By granting resource access to groups rather than to individuals, you can reduce the amount of administration required. Access Control Lists (ACLs) are based on both groups and user IDs.

Related reference

Service stanza of the qm.ini file

crtmqm (create queue manager) command

ALW Giving access to an IBM MQ object on AIX, Linux, and Windows

Use the **setmqaut** control command, the **SET AUTHREC** MQSC command, or the **MQCMD_SET_AUTH_REC** PCF command to give users, and groups of users, access to IBM MQ objects. Note that on IBM MQ Appliance you can use only the **SET AUTHREC** command.

For a full definition of the **setmqaut** control command and its syntax, see setmqaut.

For a full definition of the **SET AUTHREC** MQSC command and its syntax, see SET AUTHREC.

For a full definition of the **MQCMD_SET_AUTH_REC** PCF command and its syntax, see Set Authority Record.

The queue manager must be running to use this command. When you have changed access for a principal, the changes are reflected immediately by the OAM.

To give users access to an object, you need to specify:

- The name of the queue manager that owns the objects you are working with; if you do not specify the name of a queue manager, the default queue manager is assumed.
- The name and type of the object (to identify the object uniquely). You specify the name as a *profile* ; this is either the explicit name of the object, or a generic name, including wildcard characters. For a detailed description of generic profiles, and the use of wildcard characters within them, see "Using OAM generic profiles on AIX, Linux, and Windows" on page 348.
- One or more principals and group names to which the authority applies.

  If a user ID contains spaces, enclose it in quotation marks when you use this command. On Windows systems, you can qualify a user ID with a domain name. If the actual user ID contains an at sign (@) symbol, replace it with @@ to show that it is part of the user ID, not the delimiter between the user ID and the domain name.
- A list of authorizations. Each item in the list specifies a type of access that is to be granted to that object (or revoked from it). Each authorization in the list is specified as a keyword, prefixed with a plus sign (+) or a minus sign (-). Use a plus sign to add the specified authorization, and a minus sign to remove the authorization. There must be no spaces between the + or - sign and the keyword.

  You can specify any number of authorizations in a single command. For example, the list of authorizations to permit a user or group to put messages on a queue and to browse them, but to revoke access to get messages is:

  ```
  +browse -get +put
  ```

Examples of using the `setmqaut` command

The following examples show how to use the `setmqaut` command to grant and revoke permission to use an object:

```
setmqaut -m saturn.queue.manager -t queue -n RED.LOCAL.QUEUE
         -g groupa +browse -get +put
```

In this example:

- `saturn.queue.manager` is the queue manager name
- queue is the object type
- `RED.LOCAL.QUEUE` is the object name
- `groupa` is the identifier of the group with authorizations that are to change
- `+browse -get +put` is the authorization list for the specified queue
  - `+browse` adds authorization to browse messages on the queue (to issue **MQGET** with the browse option)
  - `-get` removes authorization to get (**MQGET**) messages from the queue
  - `+put` adds authorization to put (**MQPUT**) messages on the queue

The following command revokes put authority on the queue MyQueue from principal fvuser and from groups groupa and groupb. On AIX and Linux systems, this command also revokes put authority for all principals in the same primary group as fvuser.

```
setmqaut -m saturn.queue.manager -t queue -n MyQueue -p fvuser
         -g groupa -g groupb -put
```

Using the `setmqaut` command with a different authorization service

If you are using your own authorization service instead of the OAM, you can specify the name of this service on the **`setmqaut`** command to direct the command to this service. You must specify this parameter if you have multiple installable components running at the same time; if you do not, the update is made to the first installable component for the authorization service. By default, this is the supplied OAM.

Usage notes for SET AUTHREC

The list of authorizations to add and the list of authorizations to remove must not overlap. For example, you cannot add display authority and remove display authority with the same command. This rule applies even if the authorities are expressed using different options. For example, the following command fails because DSP authority overlaps with ALLADM authority:

```
SET AUTHREC PROFILE(*) OBJTYPE(QUEUE) PRINCIPAL(PRINC01) AUTHADD(DSP) AUTHRMV(ALLADM)
```

The exception to this overlap behavior is with the ALL authority. The following command first adds ALL authorities then removes the SETID authority:

```
SET AUTHREC PROFILE(*) OBJTYPE(QUEUE) PRINCIPAL(PRINC01) AUTHADD(ALL) AUTHRMV(SETID)
```

The following command first removes ALL authorities then adds the DSP authority:

```
SET AUTHREC PROFILE(*) OBJTYPE(QUEUE) PRINCIPAL(PRINC01) AUTHADD(DSP) AUTHRMV(ALL)
```

Regardless of the order in which they are provided on the command, the ALL are processed first.

ALW Using OAM generic profiles on AIX, Linux, and Windows

Use OAM generic profiles to set, in a single operation, a user's privileges for many objects; rather than having to issue separate **`setmqaut`** commands, or **`SET AUTHREC`** commands, against each individual object when it is created. Note that on IBM MQ Appliance you can use only the **`SET AUTHREC`** command.

Using generic profiles in the setmqaut or SET AUTHREC commands, enables you to set a generic authority for all objects that fit that profile.

This collection of topics describes the use of generic profiles in more detail.

Using wildcard characters in OAM profiles

What makes a profile generic is the use of special characters (wildcard characters) in the profile name. For example, the question mark (?) wildcard character matches any single character in a name. So, if you specify `ABC.?EF`, the authorization you give to that profile applies to any objects with the names `ABC.DEF`, `ABC.CEF`, `ABC.BEF`, and so on.

The wildcard characters available are:

?

Use the question mark (?) instead of any single character. For example, `AB.?D` applies to the objects `AB.CD`, `AB.ED`, and `AB.FD`.

*****

Use the asterisk (*) as:

- A *qualifier* in a profile name to match any one qualifier in an object name. A qualifier is the part of an object name delimited by a period. For example, in `ABC.DEF.GHI`, the qualifiers are `ABC`, `DEF`, and `GHI`.

  For example, `ABC.*.JKL` applies to the objects `ABC.DEF.JKL`, and `ABC.GHI.JKL`. (Note that it does **not** apply to `ABC.JKL` ; * used in this context always indicates one qualifier.)

- A character within a qualifier in a profile name to match zero or more characters within the qualifier in an object name.

  For example, `ABC.DE*.JKL` applies to the objects `ABC.DE.JKL`, `ABC.DEF.JKL`, and `ABC.DEGH.JKL`.

******

Use the double asterisk (**) **once** in a profile name as:

- The entire profile name to match all object names. For example if you use `-t prcs` to identify processes, then use ** as the profile name, you change the authorizations for all processes.
- As either the beginning, middle, or ending qualifier in a profile name to match zero or more qualifiers in an object name. For example, `**.ABC` identifies all objects with the final qualifier ABC.

You can only use the double asterisk ** as a complete qualifier:

```
**.DEF
  ABC.**
  A*.**
```

but not as

```
A**
```

otherwise, you receive message AMQ7226E: `The profile name is invalid.`

Note: When using wildcard characters on AIX and Linux systems, you **must** enclose the profile name in single quotation marks.

Profile priorities

An important point to understand when using generic profiles is the priority that profiles are given when deciding what authorities to apply to an object being created. For example, suppose that you have issued the commands:

```

setmqaut -n AB.* -t q +put -p fred
setmqaut -n AB.C* -t q +get -p fred
```

The first gives put authority to all queues for the principal fred with names that match the profile AB.*; the second gives get authority to the same types of queue that match the profile AB.C*.

Suppose that you now create a queue called AB.CD. According to the rules for wildcard matching, either setmqaut could apply to that queue. So, does it have put or get authority?

To find the answer, you apply the rule that, whenever multiple profiles can apply to an object, **only the most specific applies**. The way that you apply this rule is by comparing the profile names from left to right. Wherever they differ, a non-generic character is more specific then a generic character. So, in this example, the queue AB.CD has **get** authority (AB.C* is more specific than AB.*).

When you are comparing generic characters, the order of *specificity* is:

1. ?
2. *
3. **

Dumping profile settings

For a full definition of the **dmpmqaut** control command and its syntax, see dmpmqaut.

For a full definition of the **DISPLAY AUTHREC** MQSC command and its syntax, see DISPLAY AUTHREC.

For a full definition of the **MQCMD_INQUIRE_AUTH_RECS** PCF command and its syntax, see Inquire Authority Records.

The following examples show the use of the **dmpmqaut** control command to dump authority records for generic profiles:

1. This example dumps all authority records with a profile that matches queue a.b.c for principal user1.

   ```
   dmpmqaut -m qm1 -n a.b.c -t q -p user1
   ```

   The resulting dump looks something like this:

   ```
   profile:     a.b.*
   object type: queue
   entity:      user1
   type:        principal
   authority:   get, browse, put, inq
   ```

   Note: Although users on AIX and Linux can use the `-p` option for the **dmpmqaut** command, they must use `-g groupname` instead when defining authorizations.

2. This example dumps all authority records with a profile that matches queue a.b.c.

   ```
   dmpmqaut -m qmgr1 -n a.b.c -t q
   ```

   The resulting dump looks something like this:

   ```
   profile:     a.b.c
   object type: queue
   entity:      Administrator
   type:        principal
   authority:   all
   - - - - - - - - - - - - - - - - -
   profile:     a.b.*
   object type: queue
   entity:      user1
   type:        principal
   authority:   get, browse, put, inq
   - - - - - - - - - - - - - - - - -
   profile:     a.**
   object type: queue
   entity:      group1
   type:        group
   authority:   get
   ```

3. This example dumps all authority records for profile a.b.*, of type queue.

   ```
   dmpmqaut -m qmgr1 -n a.b.* -t q
   ```

   The resulting dump looks something like this:

   ```
   profile:     a.b.*
   object type: queue
   entity:      user1
   type:        principal
   authority:   get, browse, put, inq
   ```

4. This example dumps all authority records for queue manager qmX.

   ```
   dmpmqaut -m qmX
   ```

   The resulting dump looks something like this:

   ```
   profile:     q1
   object type: queue
   entity:      Administrator
   type:        principal
   authority:   all
   ```

```
- - - - - - - - - - - - - - - - -
profile:     q*
object type: queue
entity:      user1
type:        principal
authority:   get, browse
- - - - - - - - - - - - - - - - -
profile:     name.*
object type: namelist
entity:      user2
type:        principal
authority:   get
- - - - - - - - - - - - - - - - -
profile:     pr1
object type: process
entity:      group1
type:        group
authority:   get
```

5. This example dumps all profile names and object types for queue manager qmX.

   ```
   dmpmqaut -m qmX -l
   ```

   The resulting dump looks something like this:

   ```
   profile: q1, type: queue
   profile: q*, type: queue
   profile: name.*, type: namelist
   profile: pr1, type: process
   ```

Note: For IBM MQ for Windows only, all principals displayed include domain information, for example:

```
profile:     a.b.*
object type: queue
entity:      user1@domain1
type:        principal
authority:   get, browse, put, inq
```

ALW *Using wildcard characters in OAM profiles on AIX, Linux, and Windows*

Use wildcard characters in an object authority manager (OAM) profile name to make that profile applicable to more than one object.

What makes a profile generic is the use of special characters (wildcard characters) in the profile name. For example, the question mark (?) wildcard character matches any single character in a name. So, if you specify `ABC.?EF`, the authorization you give to that profile applies to any objects with the names `ABC.DEF`, `ABC.CEF`, `ABC.BEF`, and so on.

The wildcard characters available are:

?

Use the question mark (?) instead of any single character. For example, `AB.?D` applies to the objects `AB.CD`, `AB.ED`, and `AB.FD`.

*****

Use the asterisk (*) as:

- A *qualifier* in a profile name to match any one qualifier in an object name. A qualifier is the part of an object name delimited by a period. For example, in `ABC.DEF.GHI`, the qualifiers are `ABC`, `DEF`, and `GHI`.

  For example, `ABC.*.JKL` applies to the objects `ABC.DEF.JKL`, and `ABC.GHI.JKL`. (Note that it does **not** apply to `ABC.JKL` ; * used in this context always indicates one qualifier.)

- A character within a qualifier in a profile name to match zero or more characters within the qualifier in an object name.

  For example, `ABC.DE*.JKL` applies to the objects `ABC.DE.JKL`, `ABC.DEF.JKL`, and `ABC.DEGH.JKL`.

******

Use the double asterisk (**) **once** in a profile name as:

- The entire profile name to match all object names. For example if you use `-t prcs` to identify processes, then use ** as the profile name, you change the authorizations for all processes.
- As either the beginning, middle, or ending qualifier in a profile name to match zero or more qualifiers in an object name. For example, `**.`ABC identifies all objects with the final qualifier ABC.

Note: When using wildcard characters on AIX and Linux systems, you **must** enclose the profile name in single quotation marks.

ALW *Profile priorities on AIX, Linux, and Windows*

More than one generic profile can apply to a single object. Where this is the case, the most specific rule applies.

An important point to understand when using generic profiles is the priority that profiles are given when deciding what authorities to apply to an object being created. For example, suppose that you have issued the commands:

```
setmqaut -n AB.* -t q +put -p fred
setmqaut -n AB.C* -t q +get -p fred
```

The first gives put authority to all queues for the principal fred with names that match the profile AB.*; the second gives get authority to the same types of queue that match the profile AB.C*.

Suppose that you now create a queue called AB.CD. According to the rules for wildcard matching, either setmqaut could apply to that queue. So, does it have put or get authority?

To find the answer, you apply the rule that, whenever multiple profiles can apply to an object, **only the most specific applies**. The way that you apply this rule is by comparing the profile names from left to right. Wherever they differ, a non-generic character is more specific then a generic character. So, in this example, the queue AB.CD has **get** authority (AB.C* is more specific than AB.*).

When you are comparing generic characters, the order of *specificity* is:

1. ?
2. *
3. **

See SET AUTHREC for the equivalent information when using this MQSC command.

ALW *Dumping profile settings on AIX, Linux, and Windows*

Use the **`dmpmqaut`** control command, the **`DISPLAY AUTHREC`** MQSC command, or the **`MQCMD_INQUIRE_AUTH_RECS`** PCF command to dump the current authorizations associated with a specified profile. Note that on IBM MQ Appliance you can use only the **`DISPLAY AUTHREC`** command.

For a full definition of the **`dmpmqaut`** control command and its syntax, see dmpmqaut.

For a full definition of the **`DISPLAY AUTHREC`** MQSC command and its syntax, see DISPLAY AUTHREC.

For a full definition of the **`MQCMD_INQUIRE_AUTH_RECS`** PCF command and its syntax, see Inquire Authority Records.

The following examples show the use of the **`dmpmqaut`** control command to dump authority records for generic profiles:

1. This example dumps all authority records with a profile that matches queue a.b.c for principal user1.

   ```
   dmpmqaut -m qm1 -n a.b.c -t q -p user1
   ```

   The resulting dump looks something like this example:

```
profile:     a.b.*
object type: queue
entity:      user1
type:        principal
authority:   get, browse, put, inq
```

Note: AIX and Linux users cannot use the `-p` option; they must use `-g groupname` instead.

2. This example dumps all authority records with a profile that matches queue a.b.c.

```
dmpmqaut -m qmgr1 -n a.b.c -t q
```

The resulting dump looks something like this example:

```
profile:     a.b.c
object type: queue
entity:      Administrator
type:        principal
authority:   all
- - - - - - - - - - - - - - - - - -
profile:     a.b.*
object type: queue
entity:      user1
type:        principal
authority:   get, browse, put, inq
- - - - - - - - - - - - - - - - - -
profile:     a.**
object type: queue
entity:      group1
type:        group
authority:   get
```

3. This example dumps all authority records for profile a.b.*, of type queue.

```
dmpmqaut -m qmgr1 -n a.b.* -t q
```

The resulting dump looks something like this example:

```
profile:     a.b.*
object type: queue
entity:      user1
type:        principal
authority:   get, browse, put, inq
```

4. This example dumps all authority records for queue manager qmX.

```
dmpmqaut -m qmX
```

The resulting dump looks something like this example:

```
profile:     q1
object type: queue
entity:      Administrator
type:        principal
authority:   all
- - - - - - - - - - - - - - - - - -
profile:     q*
object type: queue
entity:      user1
type:        principal
authority:   get, browse
- - - - - - - - - - - - - - - - - -
profile:     name.*
object type: namelist
entity:      user2
type:        principal
authority:   get
- - - - - - - - - - - - - - - - - -
```

```
profile:     pr1
object type: process
entity:      group1
type:        group
authority:   get
```

5. This example dumps all profile names and object types for queue manager qmX.

   ```
   dmpmqaut -m qmX -l
   ```

   The resulting dump looks something like this example:

   ```
   profile: q1, type: queue
   profile: q*, type: queue
   profile: name.*, type: namelist
   profile: pr1, type: process
   ```

Note: For IBM MQ for Windows only, all principals displayed include domain information, for example:

```
profile:     a.b.*
object type: queue
entity:      user1@domain1
type:        principal
authority:   get, browse, put, inq
```

ALW Displaying access settings on AIX, Linux, and Windows

Use the **dspmqaut** control command, the **DISPLAY AUTHREC** MQSC command, or the **MQCMD_INQUIRE_ENTITY_AUTH** PCF command to view the authorizations that a specific principal or group has for a particular object. Note that on IBM MQ Appliance you can use only the **DISPLAY AUTHREC** command.

The queue manager must be running to use this command. When you change access for a principal, the changes are reflected immediately by the OAM. Authorization can be displayed for only one group or principal at a time.

For a full definition of the **dmpmqaut** control command and its syntax, see dmpmqaut.

For a full definition of the **DISPLAY AUTHREC** MQSC command and its syntax, see DISPLAY AUTHREC.

For a full definition of the **MQCMD_INQUIRE_AUTH_RECS** PCF command and its syntax, see Inquire Authority Records.

The following example shows the use of the **dspmqaut** control command to display the authorizations that the group `GpAdmin` has to a process definition named `Annuities` that is on queue manager `QueueMan1`.

```
dspmqaut -m QueueMan1 -t process -n Annuities -g GpAdmin
```

ALW Changing and revoking access to an IBM MQ object on AIX, Linux, and Windows

To change the level of access that a user or group has to an object, use the **setmqaut** control command, the **DELETE AUTHREC** MQSC command, or the **MQCMD_DELETE_AUTH_REC** PCF command. MQ Appliance Note that on IBM MQ Appliance you can use only the **DELETE AUTHREC** command.

The process of removing the user from a group is described in:

- Windows "Creating and managing groups on Windows" on page 146
- AIX "Creating and managing groups on AIX" on page 145

- Linux "Creating and managing groups on Linux" on page 145

·

The user ID that creates an IBM MQ object is granted full control authorities to that object. If you remove this user ID from the local mqm group (or the Administrators group on Windows systems) these authorities are not revoked. Use the **setmqaut** control command or the **MQCMD_DELETE_AUTH_REC** PCF command to revoke access to an object for the user ID that created it, after removing it from the mqm or Administrators group.

For a full definition of the setmqaut control command and its syntax, see setmqaut.

For a full definition of the **DELETE AUTHREC** MQSC command and its syntax, see DELETE AUTHREC.

For a full definition of the **MQCMD_DELETE_AUTH_REC** PCF command and its syntax, see Delete Authority Record.

Windows On Windows, from IBM MQ 8.0, you can delete the OAM entries corresponding to a particular Windows user account at any time using the **-u** *SID* parameter of **setmqaut**.

Prior to IBM MQ 8.0, you had to delete the OAM entries corresponding to a particular Windows user account before deleting the user profile. It was impossible to remove the OAM entries after removing the user account.

ALW Preventing security access checks on AIX, Linux, and Windows systems

Note: This topic describes functionality that is not recommended to be enabled. To turn off security checking, you can disable the object authority manager (OAM). This might be suitable for a test environment. When disabled, the queue manager is no longer be able to perform authorization or connection authentication checks. TLS, Channel Authentication records, and security exits can still be used. Having disabled or removed the OAM, you cannot add an OAM to an existing queue manager.

If you decide that you do not want to perform security checks (for example, in a test environment), you can disable the OAM in one of two ways:

- Before you create a queue manager, set the operating system environment variable **MQSNOAUT**.

  For information about the implications of setting the **MQSNOAUT** environment variable, and how you set **MQSNOAUT** on AIX, Linux, and Windows, see Environment variables descriptions.
- Edit the queue manager configuration file to remove the service.

Warning: When an OAM is removed, it cannot be put back on an existing queue manager. This is because the OAM needs to be in place at object creation time. To use the IBM MQ OAM again after it has been removed, rebuild the queue manager.

If you use the **setmqaut**, or **dspmqaut** command while the OAM is disabled, note the following points:

- The OAM does not validate the specified principal, or group, meaning that the command can accept invalid values.
- The OAM does not perform security checks and indicates that all principals and groups are authorized to perform all applicable object operations.
- Any credentials passed to the OAM for authentication checks are not validated.

Related concepts

Installable services and components for AIX, Linux, and Windows

Related tasks

Configuring installable services

Related reference

Installable services reference information

Granting required access to resources

Use this topic to determine what tasks to perform to apply security to your IBM MQ system.

About this task

During this task, you decide what actions are necessary to apply the appropriate level of security to the elements of your IBM MQ installation. Each individual task you are referred to gives step-by-step instructions for all platforms.

Procedure

1. Do you need to limit access to your queue manager to certain users?
   a) No: Take no further action.
   b) Yes: Go to the next question.
2. Do these users need partial administrative access on a subset of queue manager resources?
   a) No: Go to the next question.
   b) Yes: See "Granting partial administrative access on a subset of queue manager resources" on page 356.
3. Do these users need full administrative access on a subset of queue manager resources?
   a) No: Go to the next question.
   b) Yes: See "Granting full administrative access on a subset of queue manager resources" on page 365.
4. Do these users need read only access to all queue manager resources?
   a) No: Go to the next question.
   b) Yes: See "Granting read-only access to all resources on a queue manager" on page 371.
5. Do these users need full administrative access on all queue manager resources?
   a) No: Go to the next question.
   b) Yes: See "Granting full administrative access to all resources on a queue manager" on page 372.
6. Do you need user applications to connect to your queue manager?
   a) No: Disable connectivity, as described in "Removing connectivity to the queue manager" on page 373
   b) Yes: See "Allowing user applications to connect to your queue manager" on page 374.

Multi z/OS Granting partial administrative access on a subset of queue manager resources

You need to give certain users partial administrative access to some, but not all, queue manager resources. Use this table to determine the actions you need to take.

Table 72. Granting partial administrative access to a subset of queue manager resources

The users need to administer objects of this type	Perform this action
Queues	Grant partial administrative access to the required queues, as described in "Granting limited administrative access to some queues" on page 357
Topics	Grant partial administrative access to the required topics, as described in "Granting limited administrative access to some topics" on page 358

Table 72. Granting partial administrative access to a subset of queue manager resources (continued)

The users need to administer objects of this type	Perform this action
Channels	Grant partial administrative access to the required channels, as described in "Granting limited administrative access to some channels" on page 360
The queue manager	Grant partial administrative access to the queue manager, as described in "Granting limited administrative access to a queue manager" on page 361
Processes	Grant partial administrative access to the required processes, as described in "Granting limited administrative access to some processes" on page 362
Namelists	Grant partial administrative access to the required namelists, as described in "Granting limited administrative access to some namelists" on page 363
Services	Grant partial administrative access to the required services, as described in "Granting limited administrative access to some services" on page 364

Granting limited administrative access to some queues

Grant partial administrative access to some queues on a queue manager, to each group of users with a business need for it.

About this task

To grant limited administrative access to some queues for some actions, use the appropriate commands for your operating system.

Multi On Multiplatforms, you can also use the SET AUTHREC command.

Note: MQ Appliance On IBM MQ Appliance you can use only the **SET AUTHREC** command.

Procedure

- ALW
  For AIX, Linux, and Windows systems, issue the following command:

  ```
  setmqaut -m QMgrName -n ObjectProfile -t queue -g GroupName ReqdAction
  ```

- IBM i
  For IBM i, issue the following command:

  ```
  GRTMQMAUT OBJ(' ObjectProfile ') OBJTYPE(*Q) USER(GroupName) AUT(ReqdAction) MQMNAME('
  QMgrName ')
  ```

- z/OS For z/OS, issue the following commands to grant access to a specified queue:

```
RDEFINE MQADMIN QMgrName.QUEUE. ObjectProfile UACC(NONE)
PERMIT QMgrName.QUEUE. ObjectProfile CLASS(MQADMIN) ID(GroupName) ACCESS(ALTER)
```

To specify which MQSC commands the user can perform on the queue, issue the following commands for each MQSC command:

```
RDEFINE MQCMDS QMgrName. ReqdAction. QType UACC(NONE)
PERMIT QMgrName. ReqdAction. QType CLASS(MQCMDS) ID(GroupName) ACCESS(ALTER)
```

To permit the user to use the DISPLAY QUEUE command, issue the following commands:

```
RDEFINE MQCMDS QMgrName.DISPLAY. QType UACC(NONE)
PERMIT QMgrName.DISPLAY. QType CLASS(MQCMDS) ID(GroupName) ACCESS(READ)
```

The variable names have the following meanings:

QMgrName
The name of the queue manager.

z/OS On z/OS, this value can also be the name of a queue sharing group.

ObjectProfile
The name of the object or generic profile for which to change authorizations.

GroupName
The name of the group to be granted access.

ReqdAction
The action you are allowing the group to take:

- ALW On AIX, Linux, and Windows systems, any combination of the following authorizations: +chg, +clr, +dlt, +dsp. The authorization +alladm is equivalent to +chg +clr +dlt +dsp.
- IBM i On IBM i, any combination of the following authorizations: *ADMCHG, *ADMCLR, *ADMDLT, *ADMDSP. The authorization *ALLADM is equivalent to all these individual authorizations.
- z/OS On z/OS, one of the values ALTER, CLEAR, DELETE, or MOVE.

Note: Granting +crt for queues indirectly makes the user or group an administrator. Do not use +crt authority to grant limited administrative access to some queues.

QType

For the DISPLAY command, one of the values QUEUE, QLOCAL, QALIAS, QMODEL, QREMOTE, or QCLUSTER.

For other values of *ReqdAction*, one of the values QLOCAL, QALIAS, QMODEL, or QREMOTE.

Granting limited administrative access to some topics

Grant partial administrative access to some topics on a queue manager, to each group of users with a business need for it.

About this task

To grant limited administrative access to some topics for some actions, use the appropriate commands for your operating system.

Multi On Multiplatforms, you can also use the SET AUTHREC command.

Procedure

- ALW

  For AIX, Linux, and Windows systems, issue the following command:

  ```
  setmqaut -m QMgrName -n ObjectProfile -t topic -g GroupName ReqdAction
  ```

- IBM i

  For IBM i, issue the following command:

  ```
  GRTMQMAUT OBJ(' ObjectProfile ') OBJTYPE(*TOPIC) USER(GroupName) AUT(ReqdAction) MQMNAME('
  QMgrName ')
  ```

- z/OS For z/OS, issue the following commands:

  ```
  RDEFINE MQADMIN QMgrName.TOPIC. ObjectProfile UACC(NONE)
  PERMIT QMgrName.TOPIC. ObjectProfile CLASS(MQADMIN) ID(GroupName) ACCESS(ALTER)
  ```

  These commands grant access to the specified topic. To determine which MQSC commands the user can perform on the topic, issue the following commands for each MQSC command:

  ```
  RDEFINE MQCMDS QMgrName. ReqdAction.TOPIC UACC(NONE)
  PERMIT QMgrName. ReqdAction.TOPIC CLASS(MQCMDS) ID(GroupName) ACCESS(ALTER)
  ```

  To permit the user to use the DISPLAY TOPIC command, issue the following commands:

  ```
  RDEFINE MQCMDS QMgrName.DISPLAY.TOPIC UACC(NONE)
  PERMIT QMgrName.DISPLAY.TOPIC CLASS(MQCMDS) ID(GroupName) ACCESS(READ)
  ```

  The variable names have the following meanings:

  QMgrName
  The name of the queue manager.

  z/OS On z/OS, this value can also be the name of a queue sharing group.

  ObjectProfile
  The name of the object or generic profile for which to change authorizations.

  GroupName
  The name of the group to be granted access.

  ReqdAction
  The action you are allowing the group to take:

  - ALW On AIX, Linux, and Windows systems, any combination of the following authorizations: +chg, +clr, +crt, +dlt, +dsp. +ctrl. The authorization +alladm is equivalent to +chg +clr +dlt +dsp.
  - IBM i On IBM i, any combination of the following authorizations: *ADMCHG, *ADMCLR, *ADMCRT, *ADMDLT, *ADMDSP, *CTRL. The authorization *ALLADM is equivalent to all these individual authorizations.
  - z/OS On z/OS, one of the values ALTER, CLEAR, DEFINE, DELETE, or MOVE.

Granting limited administrative access to some channels

Grant partial administrative access to some channels on a queue manager, to each group of users with a business need for it.

About this task

To grant limited administrative access to some channels for some actions, use the appropriate commands for your operating system.

Multi On Multiplatforms, you can also use the SET AUTHREC command.

Procedure

- ALW
  On AIX, Linux, and Windows:

  ```
  setmqaut -m QMgrName -n ObjectProfile -t channel -g GroupName ReqdAction
  ```

- IBM i
  On IBM i:

  ```
  GRTMQMAUT OBJ(' ObjectProfile ') OBJTYPE(*CHL) USER(GroupName) AUT(ReqdAction) MQMNAME('
  QMgrName ')
  ```

- z/OS On z/OS:

  ```
  RDEFINE MQADMIN QMgrName.CHANNEL. ObjectProfile UACC(NONE)
  PERMIT QMgrName.CHANNEL. ObjectProfile CLASS(MQADMIN) ID(GroupName) ACCESS(ALTER)
  ```

  These commands grant access to the specified channel. To determine which MQSC commands the user can perform on the channel, issue the following commands for each MQSC command:

  ```
  RDEFINE MQCMDS QMgrName. ReqdAction.CHANNEL UACC(NONE)
  PERMIT QMgrName. ReqdAction.CHANNEL CLASS(MQCMDS) ID(GroupName) ACCESS(ALTER)
  ```

  To permit the user to use the DISPLAY CHANNEL command, issue the following commands:

  ```
  RDEFINE MQCMDS QMgrName.DISPLAY.CHANNEL UACC(NONE)
  PERMIT QMgrName.DISPLAY.CHANNEL CLASS(MQCMDS) ID(GroupName) ACCESS(READ)
  ```

  The variable names have the following meanings:

  QMgrName
  The name of the queue manager.

  z/OS On z/OS, this value can also be the name of a queue sharing group.

  ObjectProfile
  The name of the object or generic profile for which to change authorizations.

  GroupName
  The name of the group to be granted access.

  ReqdAction
  The action you are allowing the group to take:

  – ALW On AIX, Linux, and Windows, any combination of the following authorizations: +chg, +clr, +crt, +dlt, +dsp. +ctrl, +ctrlx. The authorization +alladm is equivalent to +chg +clr +dlt +dsp.

- IBM i On IBM i, any combination of the following authorizations: *ADMCHG, *ADMCLR, *ADMCRT, *ADMDLT, *ADMDSP, *CTRL, *CTRLx. The authorization *ALLADM is equivalent to all these individual authorizations.
- z/OS On z/OS, one of the values ALTER, CLEAR, DEFINE, DELETE, or MOVE.

Granting limited administrative access to a queue manager

Grant partial administrative access to a queue manager, to each group of users with a business need for it.

About this task

To grant limited administrative access to perform some actions on the queue manager, use the appropriate commands for your operating system.

Multi On Multiplatforms, you can also use the SET AUTHREC command.

Procedure

- ALW
  On AIX, Linux, and Windows:

  ```
  setmqaut -m QMgrName -n ObjectProfile -t qmgr -g GroupName ReqdAction
  ```

- IBM i
  On IBM i:

  ```
  GRTMQMAUT OBJ(' ObjectProfile ') OBJTYPE(*MQM) USER(GroupName) AUT(ReqdAction) MQMNAME('
  QMgrName ')
  ```

- z/OS
  On z/OS:

  To determine which MQSC commands you can perform on the queue manager, issue the following commands for each MQSC command:

  ```
  RDEFINE MQCMDS QMgrName. ReqdAction.QMGR UACC(NONE)
  PERMIT QMgrName. ReqdAction.QMGR CLASS(MQCMDS) ID(GroupName) ACCESS(ALTER)
  ```

  To permit the user to use the DISPLAY QMGR command, issue the following commands:

  ```
  RDEFINE MQCMDS QMgrName.DISPLAY.QMGR UACC(NONE)
  PERMIT QMgrName.DISPLAY.QMGR CLASS(MQCMDS) ID(GroupName) ACCESS(READ)
  ```

  The variable names have the following meanings:

  QMgrName
  : The name of the queue manager.

  ObjectProfile
  : The name of the object or generic profile for which to change authorizations.

  GroupName
  : The name of the group to be granted access.

  ReqdAction
  : The action you are allowing the group to take:

  - ALW On AIX, Linux, and Windows, any combination of the following authorizations: +chg, +clr, +crt, +dlt, +dsp. The authorization +alladm is equivalent to +chg +clr +dlt +dsp.

Although +set is an MQI authorization and not normally considered administrative, granting +set on the queue manager can indirectly lead to full administrative authority. Do not grant +set to ordinary users and applications.

  – IBM i On IBM i, any combination of the following authorizations: *ADMCHG, *ADMCLR, *ADMCRT, *ADMDLT, *ADMDSP. The authorization *ALLADM is equivalent to all these individual authorizations.

Granting limited administrative access to some processes

Grant partial administrative access to some processes on a queue manager, to each group of users with a business need for it.

About this task

To grant limited administrative access to some processes for some actions, use the appropriate commands for your operating system.

Multi On Multiplatforms, you can also use the SET AUTHREC command.

Procedure

- ALW
  On AIX, Linux, and Windows:

```
setmqaut -m QMgrName -n ObjectProfile -t process -g GroupName ReqdAction
```

- IBM i
  On IBM i:

```
GRTMQMAUT OBJ(' ObjectProfile ') OBJTYPE(*PRC) USER(GroupName) AUT(ReqdAction) MQMNAME('
QMgrName ')
```

- z/OS On z/OS:

```
RDEFINE MQADMIN QMgrName.PROCESS. ObjectProfile UACC(NONE)
PERMIT QMgrName.PROCESS. ObjectProfile CLASS(MQADMIN) ID(GroupName) ACCESS(ALTER)
```

  These commands grant access to the specified channel. To determine which MQSC commands the user can perform on the channel, issue the following commands for each MQSC command:

```
RDEFINE MQCMDS QMgrName. ReqdAction.PROCESS UACC(NONE)
PERMIT QMgrName. ReqdAction.PROCESS CLASS(MQCMDS) ID(GroupName) ACCESS(ALTER)
```

  To permit the user to use the DISPLAY PROCESS command, issue the following commands:

```
RDEFINE MQCMDS QMgrName.DISPLAY.PROCESS UACC(NONE)
PERMIT QMgrName.DISPLAY.PROCESS CLASS(MQCMDS) ID(GroupName) ACCESS(READ)
```

  The variable names have the following meanings:

  QMgrName
    The name of the queue manager.

    z/OS On z/OS, this value can also be the name of a queue sharing group.

  ObjectProfile
    The name of the object or generic profile for which to change authorizations.

GroupName
The name of the group to be granted access.

ReqdAction
The action you are allowing the group to take:

- ALW On AIX, Linux, and Windows, any combination of the following authorizations: +chg, +clr, +crt, +dlt, +dsp. The authorization +alladm is equivalent to +chg +clr +dlt +dsp.
- IBM i On IBM i, any combination of the following authorizations: *ADMCHG, *ADMCLR, *ADMCRT, *ADMDLT, *ADMDSP. The authorization *ALLADM is equivalent to all these individual authorizations.
- z/OS On z/OS, one of the values ALTER, CLEAR, DEFINE, DELETE, or MOVE.

Granting limited administrative access to some namelists

Grant partial administrative access to some namelists on a queue manager, to each group of users with a business need for it.

About this task

To grant limited administrative access to some namelists for some actions, use the appropriate commands for your operating system.

Multi On Multiplatforms, you can also use the SET AUTHREC command.

Procedure

- ALW
  On AIX, Linux, and Windows:

```
setmqaut -m QMgrName -n ObjectProfile -t namelist -g GroupName ReqdAction
```

- IBM i
  On IBM i:

```
GRTMQMAUT OBJ(' ObjectProfile ') OBJTYPE(*NMLIST) USER(GroupName) AUT(ReqdAction) MQMNAME('
QMgrName ')
```

- z/OS On z/OS:

```
RDEFINE MQADMIN QMgrName.NAMELIST. ObjectProfile UACC(NONE)
PERMIT QMgrName.NAMELIST. ObjectProfile CLASS(MQADMIN) ID(GroupName) ACCESS(ALTER)
```

These commands grant access to the specified namelist. To determine which MQSC commands the user can perform on the namelist, issue the following commands for each MQSC command:

```
RDEFINE MQCMDS QMgrName. ReqdAction.NAMELIST UACC(NONE)
PERMIT QMgrName. ReqdAction.NAMELIST CLASS(MQCMDS) ID(GroupName) ACCESS(ALTER)
```

To permit the user to use the DISPLAY NAMELIST command, issue the following commands:

```
RDEFINE MQCMDS QMgrName.DISPLAY.NAMELIST UACC(NONE)
PERMIT QMgrName.DISPLAY.NAMELIST CLASS(MQCMDS) ID(GroupName) ACCESS(READ)
```

The variable names have the following meanings:

QMgrName
The name of the queue manager.

z/OS On z/OS, this value can also be the name of a queue sharing group.

ObjectProfile
The name of the object or generic profile for which to change authorizations.

GroupName
The name of the group to be granted access.

ReqdAction
The action you are allowing the group to take:

- ALW On AIX, Linux, and Windows, any combination of the following authorizations: +chg, +clr, +crt, +dlt, +ctrl, +ctrlx, +dsp. The authorization +alladm is equivalent to +chg +clr +dlt +dsp.
- IBM i On IBM i, any combination of the following authorizations: *ADMCHG, *ADMCLR, *ADMCRT, *ADMDLT, *ADMDSP, *CTRL, *CTRLX. The authorization *ALLADM is equivalent to all these individual authorizations.
- z/OS On z/OS, one of the values ALTER, CLEAR, DEFINE, DELETE, or MOVE.

Granting limited administrative access to some services

Grant partial administrative access to some services on a queue manager, to each group of users with a business need for it.

About this task

To grant limited administrative access to some services for some actions, use the appropriate commands for your operating system. z/OS Note that service objects do not exist on z/OS.

Multi On Multiplatforms, you can also use the SET AUTHREC command.

Procedure

- ALW
  On AIX, Linux, and Windows:

```
setmqaut -m QMgrName -n ObjectProfile -t service -g GroupName ReqdAction
```

- On IBM i:

```
GRTMQMAUT OBJ(' ObjectProfile ') OBJTYPE(*SVC) USER(GroupName) AUT(ReqdAction) MQMNAME('
QMgrName ')
```

- z/OS On z/OS:

  These commands grant access to the specified service. To determine which MQSC commands the user can perform on the service, issue the following commands for each MQSC command:

```
RDEFINE MQCMDS QMgrName. ReqdAction.SERVICE UACC(NONE)
PERMIT QMgrName. ReqdAction.SERVICE CLASS(MQCMDS) ID(GroupName) ACCESS(ALTER)
```

  To permit the user to use the DISPLAY SERVICE command, issue the following commands:

```
RDEFINE MQCMDS QMgrName.DISPLAY.SERVICE UACC(NONE)
PERMIT QMgrName.DISPLAY.SERVICE CLASS(MQCMDS) ID(GroupName) ACCESS(READ)
```

  The variable names have the following meanings:

QMgrName
The name of the queue manager.

ObjectProfile
The name of the object or generic profile for which to change authorizations.

GroupName
The name of the group to be granted access.

ReqdAction
The action you are allowing the group to take:

- ALW On AIX, Linux, and Windows systems, any combination of the following authorizations: +chg, +clr, +crt, +dlt, +ctrl, +ctrlx, +dsp. The authorization +alladm is equivalent to +chg +clr +dlt +dsp.
- IBM i On IBM i, any combination of the following authorizations: *ADMCHG, *ADMCLR, *ADMCRT, *ADMDLT, *ADMDSP, *CTRL, *CTRLX. The authorization *ALLADM is equivalent to all these individual authorizations.

Granting full administrative access on a subset of queue manager resources

You need to give certain users full administrative access to some, but not all, queue manager resources. Use these tables to determine the actions you need to take.

Table 73. Granting full administrative access to a subset of queue manager resources

The users need to administer objects of this type	Perform this action
Queues	Grant full administrative access to the required queues, as described in "Granting full administrative access to some queues" on page 366
Topics	Grant full administrative access to the required topics, as described in "Granting full administrative access to some topics" on page 366
Channels	Grant full administrative access to the required channels, as described in "Granting full administrative access to some channels" on page 367
The queue manager	Grant full administrative access to the queue manager, as described in "Granting full administrative access to a queue manager" on page 368
Processes	Grant full administrative access to the required processes, as described in "Granting full administrative access to some processes" on page 369
Namelists	Grant full administrative access to the required namelists, as described in "Granting full administrative access to some namelists" on page 369
Services	Grant full administrative access to the required services, as described in "Granting full administrative access to some services" on page 370

Granting full administrative access to some queues
Grant full administrative access to some queues on a queue manager, to each group of users with a business need for it.

About this task
To grant full administrative access to some queues, use the appropriate commands for your operating system.

Multi On Multiplatforms, you can also use the SET AUTHREC command.

Procedure
- ALW
  On AIX, Linux, and Windows:

  ```
  setmqaut -m QMgrName -n ObjectProfile -t queue -g GroupName +alladm
  ```
- IBM i
  On IBM i:

  ```
  GRTMQMAUT OBJ(' ObjectProfile ') OBJTYPE(*Q) USER(GroupName) AUT(*ALLADM) MQMNAME(' QMgrName
  ')
  ```
- z/OS
  On z/OS:

  ```
  RDEFINE MQADMIN QMgrName.QUEUE. ObjectProfile UACC(NONE)
  PERMIT QMgrName.QUEUE. ObjectProfile CLASS(MQADMIN) ID(GroupName) ACCESS(ALTER)
  ```

  The variable names have the following meanings:

  QMgrName
  The name of the queue manager.

  z/OS On z/OS, this value can also be the name of a queue sharing group.

  ObjectProfile
  The name of the object or generic profile for which to change authorizations.

  GroupName
  The name of the group to be granted access.

Granting full administrative access to some topics
Grant full administrative access to some topics on a queue manager, to each group of users with a business need for it.

About this task
To grant full administrative access to some topics for some actions, use the appropriate commands for your operating system.

Multi On Multiplatforms, you can also use the SET AUTHREC command.

Procedure
- ALW
  On AIX, Linux, and Windows:

```
setmqaut -m QMgrName -n ObjectProfile -t topic -g GroupName +alladm
```

- IBM i
  On IBM i:

  ```
  GRTMQMAUT OBJ(' ObjectProfile ') OBJTYPE(*TOPIC) USER(GroupName) AUT(ALLADM) MQMNAME('
  QMgrName ')
  ```

- z/OS
  On z/OS:

  ```
  RDEFINE MQADMIN QMgrName.TOPIC. ObjectProfile UACC(NONE)
  PERMIT QMgrName.TOPIC. ObjectProfile CLASS(MQADMIN) ID(GroupName) ACCESS(ALTER)
  ```

  The variable names have the following meanings:

  QMgrName
  The name of the queue manager.

  z/OS On z/OS, this value can also be the name of a queue sharing group.

  ObjectProfile
  The name of the object or generic profile for which to change authorizations.

  GroupName
  The name of the group to be granted access.

Granting full administrative access to some channels

Grant full administrative access to some channels on a queue manager, to each group of users with a business need for it.

About this task

To grant full administrative access to some channels, use the appropriate commands for your operating system.

Multi On Multiplatforms, you can also use the SET AUTHREC command.

Procedure

- ALW
  On AIX, Linux, and Windows:

  ```
  setmqaut -m QMgrName -n ObjectProfile -t channel -g GroupName +alladm
  ```

- IBM i
  On IBM i:

  ```
  GRTMQMAUT OBJ(' ObjectProfile ') OBJTYPE(*CHL) USER(GroupName) AUT(ALLADM) MQMNAME('
  QMgrName ')
  ```

- z/OS
  On z/OS:

  ```
  RDEFINE MQADMIN QMgrName.CHANNEL. ObjectProfile UACC(NONE)
  PERMIT QMgrName.CHANNEL. ObjectProfile CLASS(MQADMIN) ID(GroupName) ACCESS(ALTER)
  ```

The variable names have the following meanings:

QMgrName
: The name of the queue manager.

  z/OS On z/OS, this value can also be the name of a queue sharing group.

ObjectProfile
: The name of the object or generic profile for which to change authorizations.

GroupName
: The name of the group to be granted access.

Granting full administrative access to a queue manager

Grant full administrative access to a queue manager, to each group of users with a business need for it.

About this task

To grant full administrative access to the queue manager, use the appropriate commands for your operating system.

Multi On Multiplatforms, you can also use the SET AUTHREC command.

Procedure

- ALW
  On AIX, Linux, and Windows:

  ```
  setmqaut -m QMgrName -t qmgr -g GroupName +alladm
  ```

- IBM i
  On IBM i:

  ```
  GRTMQMAUT OBJ(' ObjectProfile ') OBJTYPE(*MQM) USER(GroupName) AUT(*ALLADM) MQMNAME('
  QMgrName ')
  ```

- z/OS
  On z/OS:

  ```
  RDEFINE MQADMIN QMgrName.QMGR UACC(NONE)
  PERMIT QMgrName.QMGR CLASS(MQADMIN) ID(GroupName) ACCESS(ALTER)
  ```

  The variable names have the following meanings:

  QMgrName
  : The name of the queue manager.

    z/OS On z/OS, this value can also be the name of a queue sharing group.

  ObjectProfile
  : The name of the object or generic profile for which to change authorizations.

  GroupName
  : The name of the group to be granted access.

Granting full administrative access to some processes

Grant full administrative access to some processes on a queue manager, to each group of users with a business need for it.

About this task

To grant full administrative access to some processes, use the appropriate commands for your operating system.

Multi On Multiplatforms, you can also use the SET AUTHREC command.

Procedure

- ALW

  On AIX, Linux, and Windows:

  ```
  setmqaut -m QMgrName -n ObjectProfile -t process -g GroupName +alladm
  ```

- IBM i

  On IBM i:

  ```
  GRTMQMAUT OBJ(' ObjectProfile ') OBJTYPE(*PRC) USER(GroupName) AUT(*ALLADM) MQMNAME('
  QMgrName ')
  ```

- z/OS

  On z/OS:

  ```
  RDEFINE MQADMIN QMgrName.CHANNEL. ObjectProfile UACC(NONE)
  PERMIT QMgrName.PROCESS. ObjectProfile CLASS(MQADMIN) ID(GroupName) ACCESS(ALTER)
  ```

  The variable names have the following meanings:

  QMgrName
  : The name of the queue manager.

    z/OS On z/OS, this value can also be the name of a queue sharing group.

  ObjectProfile
  : The name of the object or generic profile for which to change authorizations.

  GroupName
  : The name of the group to be granted access.

Granting full administrative access to some namelists

Grant full administrative access to some namelists on a queue manager, to each group of users with a business need for it.

About this task

To grant full administrative access to some namelists, use the appropriate commands for your operating system.

Multi On Multiplatforms, you can also use the SET AUTHREC command.

Procedure

- ALW

  On AIX, Linux, and Windows:

```
setmqaut -m QMgrName -n ObjectProfile -t namelist -g GroupName +alladm
```

- IBM i

  On IBM i:

  ```
  GRTMQMAUT OBJ(' ObjectProfile ') OBJTYPE(*NMLIST) USER(GroupName) AUT(*ALLADM) MQMNAME('
  QMgrName ')
  ```

- z/OS

  On z/OS:

  ```
  RDEFINE MQADMIN QMgrName.NAMELIST. ObjectProfile UACC(NONE)
  PERMIT QMgrName.NAMELIST. ObjectProfile CLASS(MQADMIN) ID(GroupName) ACCESS(ALTER)
  ```

The variable names have the following meanings:

QMgrName
The name of the queue manager.

z/OS On z/OS, this value can also be the name of a queue sharing group.

ObjectProfile
The name of the object or generic profile for which to change authorizations.

GroupName
The name of the group to be granted access.

Granting full administrative access to some services

Grant full administrative access to some services on a queue manager, to each group of users with a business need for it.

About this task

To grant full administrative access to some services, use the appropriate commands for your operating system.

Multi On Multiplatforms, you can also use the SET AUTHREC command.

Procedure

- ALW

  On AIX, Linux, and Windows:

  ```
  setmqaut -m QMgrName -n ObjectProfile -t service -g GroupName +alladm
  ```

- IBM i

  On IBM i:

  ```
  GRTMQMAUT OBJ(' ObjectProfile ') OBJTYPE(*SVC) USER(GroupName) AUT(*ALLADM) MQMNAME('
  QMgrName ')
  ```

- z/OS

  On z/OS:

  ```
  RDEFINE MQADMIN QMgrName.SERVICE. ObjectProfile UACC(NONE)
  PERMIT QMgrName.SERVICE. ObjectProfile CLASS(MQADMIN) ID(GroupName) ACCESS(ALTER)
  ```

The variable names have the following meanings:

QMgrName
: The name of the queue manager.

z/OS On z/OS, this value can also be the name of a queue sharing group.

ObjectProfile
: The name of the object or generic profile for which to change authorizations.

GroupName
: The name of the group to be granted access.

Granting read-only access to all resources on a queue manager

Grant read-only access to all the resources on a queue manager, to each user or group of users with a business need for it.

About this task

Use the Add Role Based Authorities wizard or the appropriate commands for your operating system.

Multi On Multiplatforms, you can also use the SET AUTHREC command.

After you have changed any authorization details perform a security refresh using the REFRESH SECURITY command.

Procedure

- Using the wizard:
  a) In the IBM MQ Explorer Navigator pane, right-click the queue manager and click **Object Authorities** > **Add Role Based Authorities**

     The Add Role Based Authorities wizard opens.
- ALW

  For AIX, Linux, and Windows systems, issue the following commands:

```
setmqaut -m QMgrName -n ** -t queue -g GroupName +browse +dsp
setmqaut -m QMgrName -n SYSTEM.ADMIN.COMMAND.QUEUE -t queue -g GroupName +dsp +inq +put
setmqaut -m QMgrName -n SYSTEM.MQEXPLORER.REPLY.MODEL -t queue -g GroupName +dsp +inq +get
+put
setmqaut -m QMgrName -n ** -t topic -g GroupName +dsp
setmqaut -m QMgrName -n ** -t channel -g GroupName +dsp
setmqaut -m QMgrName -n ** -t clntconn -g GroupName +dsp
setmqaut -m QMgrName -n ** -t authinfo -g GroupName +dsp
setmqaut -m QMgrName -n ** -t listener -g GroupName +dsp
setmqaut -m QMgrName -n ** -t namelist -g GroupName +dsp
setmqaut -m QMgrName -n ** -t process -g GroupName +dsp
setmqaut -m QMgrName -n ** -t service -g GroupName +dsp
setmqaut -m QMgrName -t qmgr -g GroupName +dsp +inq +connect
```

  The specific authorities to SYSTEM.ADMIN.COMMAND.QUEUE and SYSTEM.MQEXPLORER.REPLY.MODEL are necessary only if you want to use the IBM MQ Explorer.
- IBM i

  For IBM i, issue the following commands:

```
GRTMQMAUT OBJ(*ALL) OBJTYPE(*Q) USER('GroupName') AUT(*ADMDSP *BROWSE) MQMNAME('QMgrName')
GRTMQMAUT OBJ(*ALL) OBJTYPE(*TOPIC) USER('GroupName') AUT(*ADMDSP) MQMNAME('QMgrName')
GRTMQMAUT OBJ(*ALL) OBJTYPE(*CHL) USER('GroupName') AUT(*ADMDSP *INQ) MQMNAME('QMgrName')
GRTMQMAUT OBJ(*ALL) OBJTYPE(*CLTCN) USER('GroupName') AUT(*ADMDSP) MQMNAME('QMgrName')
GRTMQMAUT OBJ(*ALL) OBJTYPE(*AUTHINFO) USER('GroupName') AUT(*ADMDSP) MQMNAME('QMgrName')
GRTMQMAUT OBJ(*ALL) OBJTYPE(*LSR) USER('GroupName') AUT(*ADMDSP)MQMNAME('QMgrName')
GRTMQMAUT OBJ(*ALL) OBJTYPE(*NMLIST) USER('GroupName') AUT(*ADMDSP) MQMNAME('QMgrName')
GRTMQMAUT OBJ(*ALL) OBJTYPE(*PRC) USER('GroupName') AUT(*ADMDSP) MQMNAME('QMgrName')
GRTMQMAUT OBJ(*ALL) OBJTYPE(*SVC) USER('GroupName') AUT(*ADMDSP) MQMNAME('QMgrName')
GRTMQMAUT OBJ('object-name') OBJTYPE(*MQM) USER('GroupName') AUT(*ADMDSP *CONNECT *INQ)
MQMNAME('QMgrName')
```

- z/OS

  For z/OS, issue the following commands:

  ```
  RDEFINE MQQUEUE QMgrName.** UACC(NONE)
  PERMIT QMgrName.** CLASS(MQQUEUE) ID(GroupName) ACCESS(READ)
  RDEFINE MXTOPIC QMgrName.** UACC(NONE)
  PERMIT QMgrName.** CLASS(MXTOPIC) ID(GroupName) ACCESS(READ)
  RDEFINE MQPROC QMgrName.** UACC(NONE)
  PERMIT QMgrName.** CLASS(MQPROC) ID(GroupName) ACCESS(READ)
  RDEFINE MQNLIST QMgrName.** UACC(NONE)
  PERMIT QMgrName.** CLASS(MQNLIST) ID(GroupName) ACCESS(READ)
  RDEFINE MQCONN QMgrName.BATCH UACC(NONE)
  PERMIT QMgrName.BATCH CLASS(MQCONN) ID(GroupName) ACCESS(READ)
  RDEFINE MQCONN QMgrName.CICS UACC(NONE)
  PERMIT QMgrName.CICS CLASS(MQCONN) ID(GroupName) ACCESS(READ)
  RDEFINE MQCONN QMgrName.IMS UACC(NONE)
  PERMIT QMgrName.IMS CLASS(MQCONN) ID(GroupName) ACCESS(READ)
  RDEFINE MQCONN QMgrName.CHIN UACC(NONE)
  PERMIT QMgrName.CHIN CLASS(MQCONN) ID(GroupName) ACCESS(READ)
  ```

  The variable names have the following meanings:

  QMgrName
  : The name of the queue manager.

    z/OS On z/OS, this value can also be the name of a queue sharing group.

  GroupName
  : The name of the group to be granted access.

Granting full administrative access to all resources on a queue manager

Grant full administrative access to all the resources on a queue manager, to each user or group of users with a business need for it.

About this task

You can use the Add Role Based Authorities wizard or the appropriate commands for your operating system.

Multi On Multiplatforms, you can also use the SET AUTHREC command.

Notes: ALW

1. If you are using **runmqsc** to administer the queue manager instead of the IBM MQ Explorer, you must grant authority to inquire, get, and browse the SYSTEM.MQSC.REPLY.QUEUE, and you do not need to grant any authorities on the SYSTEM.MQEXPLORER.REPLY.MODEL queue.
2. When giving a user access to all resources on a queue manager there are some commands that the user cannot run, unless that user has read access to the `qm.ini` file. This is due to restrictions on non mqm users being able to read the `qm.ini` file.

   The user cannot issue the following commands unless you have granted that user read access to the qm.ini file:

   - Defining a channel that is configured to use TLS
   - Defining a channel using auto-configuration insertion variables defined in `qm.ini`

Procedure

- If you are using the wizard, in the IBM MQ Explorer Navigator pane, right-click the queue manager and click **Object Authorities** > **Add Role Based Authorities**.

  The Add Role Based Authorities wizard opens.

- Linux AIX
  For AIX and Linux systems, issue the following commands:

```
setmqaut -m QMgrName -n '**' -t queue -g GroupName +alladm +browse
setmqaut -m QMgrName -n @class -t queue -g GroupName +crt
setmqaut -m QMgrName -n SYSTEM.ADMIN.COMMAND.QUEUE -t queue -g GroupName +dsp +inq +put
setmqaut -m QMgrName -n SYSTEM.MQEXPLORER.REPLY.MODEL -t queue -g GroupName +dsp +inq +get +put
setmqaut -m QMgrName -n '**' -t topic -g GroupName +alladm
setmqaut -m QMgrName -n @class -t topic -g GroupName +crt
setmqaut -m QMgrName -n '**' -t channel -g GroupName +alladm
setmqaut -m QMgrName -n @class -t channel -g GroupName +crt
setmqaut -m QMgrName -n '**' -t clntconn -g GroupName +alladm
setmqaut -m QMgrName -n @class -t clntconn -g GroupName +crt
setmqaut -m QMgrName -n '**' -t authinfo -g GroupName +alladm
setmqaut -m QMgrName -n @class -t authinfo -g GroupName +crt
setmqaut -m QMgrName -n '**' -t listener -g GroupName +alladm
setmqaut -m QMgrName -n @class -t listener -g GroupName +crt
setmqaut -m QMgrName -n '**' -t namelist -g GroupName +alladm
setmqaut -m QMgrName -n @class -t namelist -g GroupName +crt
setmqaut -m QMgrName -n '**' -t process -g GroupName +alladm
setmqaut -m QMgrName -n @class -t process -g GroupName +crt
setmqaut -m QMgrName -n '**' -t service -g GroupName +alladm
setmqaut -m QMgrName -n @class -t service -g GroupName +crt
setmqaut -m QMgrName -t qmgr -g GroupName +alladm +connect
```

  See **`setmqaut`** for more information on `@class`

- Windows
  For Windows systems, issue the same commands as for AIX and Linux systems, but using the profile name @CLASS instead of @class.

- IBM i
  For IBM i, issue the following command:

```
GRTMQMAUT OBJ(*ALL) OBJTYPE(*ALL) USER(' GroupName ') AUT(*ALLADM) MQMNAME(' QMgrName ')
```

- z/OS
  For z/OS, issue the following commands:

```
RDEFINE MQADMIN QMgrName.*.** UACC(NONE)
PERMIT QMgrName.*.** CLASS(MQADMIN) ID(GroupName) ACCESS(ALTER)
```

  The variable names have the following meanings:

  QMgrName
  The name of the queue manager.

  z/OS On z/OS, this value can also be the name of a queue sharing group.

  GroupName
  The name of the group to be granted access.

Removing connectivity to the queue manager

If you do not want user applications to connect to your queue manager, remove their authority to connect to it.

About this task

Revoke the authority of all users to connect to the queue manager by using the appropriate command for your operating system.

On Multiplatforms, you can also use the DELETE AUTHREC command.

Note: On IBM MQ Appliance you can use only the **DELETE AUTHREC** command.

Procedure

- ALW

  For AIX, Linux, and Windows systems, issue the following command:

  ```
  setmqaut -m QMgrName -t qmgr -g GroupName -connect
  ```

- IBM i

  For IBM i, issue the following command:

  ```
  RVKMQMAUT OBJ ('QMgrName') OBJTYPE(*MQM) USER(*ALL) AUT(*CONNECT)
  ```

- z/OS

  For z/OS, issue the following commands:

  ```
  RDEFINE MQCONN QMgrName.BATCH UACC(NONE)
  RDEFINE MQCONN QMgrName.CHIN UACC(NONE)
  RDEFINE MQCONN QMgrName.CICS UACC(NONE)
  RDEFINE MQCONN QMgrName.IMS UACC(NONE)
  ```

  Do not issue any PERMIT commands.

  The variable names have the following meanings:

  QMgrName
  The name of the queue manager.

  z/OS On z/OS, this value can also be the name of a queue sharing group.

  GroupName
  The name of the group to be denied access.

Allowing user applications to connect to your queue manager

You want to allow user application to connect to your queue manager. Use the tables in this topic to determine what actions to take.

First, determine whether client applications will connect to your queue manager.

If none of the applications that will connect to your queue manager are client applications, disable remote access as described in "Disabling remote access to the queue manager" on page 382.

If one or more of the applications that will connect to your queue manager are client applications, secure remote connectivity as described in "Securing remote connectivity to the queue manager" on page 375.

In both cases, set up connection security as described in "Setting up connection security" on page 382

If you want to control access to resources for each user connecting to the queue manager, see the following table. If the statement in the first column is true, take the action listed in the second column.

Statement	Take this action
You have applications that make use of queues	See "Controlling user access to queues" on page 383
You have applications that make use of topics	See "Controlling user access to topics" on page 388.
You have applications that inquire on the queue manager object	See "Granting authority to inquire on a queue manager" on page 390.
You have applications that use process objects	See "Granting authority to access processes" on page 391

Statement	Take this action
You have applications that make use of namelists	See “Granting authority to access namelists” on page 392

Securing remote connectivity to the queue manager

You can secure remote connectivity to the queue manager using TLS, a security exit, channel authentication records, or a combination of these methods.

About this task

You connect a client to the queue manager by using a client-connection channel on the client workstation and a server-connection channel on the server. Secure such connections in one of the following ways.

Procedure

1. Using TLS with channel authentication records:
   a) Prevent any Distinguished Name (DN) from opening a channel, by using an SSLPEERMAP channel authentication record to map all DNs to USERSRC(NOACCESS).
   b) Allow specific DNs or sets of DNs to open a channel by using an SSLPEERMAP channel authentication record to map them to USERSRC(CHANNEL).
2. Using TLS with a security exit:
   a) Set MCAUSER on the server-connection channel to a user identifier with no privileges.
   b) Write a security exit to assign an MCAUSER value depending on the value of TLS DN it receives in the SSLPeerNamePtr and SSLPeerNameLength fields passed to the exit in the MQCD structure.
3. Using TLS with fixed channel definition values:
   a) Set SSLPEER on the server-connection channel to a specific value or narrow range of values.
   b) Set MCAUSER on the server-connection channel to the user ID the channel should run with.
4. Using channel authentication records on channels that do not use TLS:
   a) Prevent any IP address from opening channels, by using an address-mapping channel authentication record with ADDRESS(*) and USERSRC(NOACCESS).
   b) Allow specific IP addresses to open channels, by using address-mapping channel authentication records for those addresses with USERSRC(CHANNEL).
5. Using a security exit:
   a) Write a security exit to authorize connections based on any property you choose, for example, the originating IP address.
6. It is also possible to use channel authentication records with a security exit, or to use all three methods, if your particular circumstances require it.

Blocking specific IP addresses

You can prevent a specific channel accepting an inbound connection from an IP address, or prevent the whole queue manager from allowing access from an IP address, by using a channel authentication record.

Before you begin

Enable channel authentication records by running the following command:

```
ALTER QMGR CHLAUTH(ENABLED)
```

About this task

To disallow specific channels from accepting an inbound connection and ensure that connections are only accepted when using the correct channel name, one type of rule can be used to block IP addresses.
To disallow an IP address access to the whole queue manager, you would normally use a firewall to

permanently block it. However, another type of rule can be used to allow you to block a few addresses temporarily, for example while you are waiting for the firewall to be updated.

Procedure

- To block IP addresses from using a specific channel, set a channel authentication record by using the MQSC command **`SET CHLAUTH`**, or the PCF command **`Set Channel Authentication Record`**.

```
SET CHLAUTH(generic-channel-name) TYPE(ADDRESSMAP) ADDRESS(generic-ip-address)
USERSRC(NOACCESS)
```

There are three parts to the command:

SET CHLAUTH (*generic-channel-name*)
You use this part of the command to control whether you want to block a connection for the entire queue manager, single channel or range of channels. What you put in here determines which areas are covered.

For example:

- `SET CHLAUTH('*')` - blocks every channel on a queue manager, that is, the entire queue manager
- SET CHLAUTH('SYSTEM.*') - blocks every channel that begins with SYSTEM.
- SET CHLAUTH('SYSTEM.DEF.SVRCONN') - blocks the channel SYSTEM.DEF.SVRCONN

Type of CHLAUTH rule
Use this part of the command to specify the type of command and determines whether you want to supply a single address or list of addresses.

For example:

- `TYPE(ADDRESSMAP)` - Use ADDRESSMAP if you want to supply a single address or wildcard address. For example, `ADDRESS('192.168.*')` blocks any connections coming from an IP address starting in `192.168`.

  For more information about filtering IP addresses with patterns, see Generic IP addresses.
- `TYPE(BLOCKADDR)` - Use BLOCKADDR if you want to supply a list of address to block.

Additional parameters
These parameters are dependent upon the type of rule you used in the second part of the command:

- For `TYPE(ADDRESSMAP)` you use ADDRESS
- For `TYPE(BLOCKADDR)` you use ADDRLIST

Related reference
SET CHLAUTH

Temporarily blocking specific IP addresses if the queue manager is not running
You might want to block particular IP addresses, or ranges of addresses, when the queue manager is not running and you cannot therefore issue MQSC commands. You can temporarily block IP addresses on an exceptional basis by modifying the `blockaddr.ini` file.

About this task

The `blockaddr.ini` file contains a copy of the BLOCKADDR definitions that are used by the queue manager. This file is read by the listener if the listener is started before the queue manager. In these circumstances, the listener uses any values that you have manually added to the `blockaddr.ini` file.

However, be aware that when the queue manager is started, it writes the set of BLOCKADDR definitions to the `blockaddr.ini` file, over-writing any manual editing you might have done. Similarly, every time you add or delete a BLOCKADDR definition by using the **`SET CHLAUTH`** command, the `blockaddr.ini` file is updated. You can therefore make permanent changes to the BLOCKADDR definitions only by using the **SET CHLAUTH** command when the queue manager is running.

Procedure

1. Open the `blockaddr.ini` file in a text editor.

   The file is located in the data directory of the queue manager.
2. Add IP addresses as simple keyword-value pairs, where the keyword is `Addr`.

   For information about filtering IP addresses with patterns, see Generic IP addresses.

   For example:

   ```
   Addr = 192.0.2.0
   Addr = 192.0.*
   Addr = 192.0.2.1-8
   ```

Related tasks

"Blocking specific IP addresses" on page 375
You can prevent a specific channel accepting an inbound connection from an IP address, or prevent the whole queue manager from allowing access from an IP address, by using a channel authentication record.

Related reference

SET CHLAUTH

Blocking specific user IDs
You can prevent specific users from using a channel by specifying user IDs that, if asserted, cause the channel to end. Do this by setting a channel authentication record.

Before you begin

Ensure that channel authentication records are enabled as follows:

```
ALTER QMGR CHLAUTH(ENABLED)
```

Procedure

Set a channel authentication record using the MQSC command **SET CHLAUTH**, or the PCF command **Set Channel Authentication Record**. For example, you can issue the MQSC command:

```
SET CHLAUTH(' generic-channel-name ') TYPE(BLOCKUSER) USERLIST(userID1, userID2)
```

generic-channel-name is either the name of a channel to which you want to control access, or a pattern including the asterisk (*) symbol as a wildcard that matches the channel name.

The user list provided on a `TYPE(BLOCKUSER)` only applies to SVRCONN channels and not queue manager to queue manager channels.

userID1 and *userID2* are each the ID of a user that is to be prevented from using the channel. You can also specify the special value `*MQADMIN` to refer to privileged administrative users. For more information about privileged users, see "Privileged users" on page 313. For more information about `*MQADMIN`, see SET CHLAUTH.

Related reference

SET CHLAUTH

Mapping a remote queue manager to an MCAUSER user ID
You can use a channel authentication record to set the MCAUSER attribute of a channel, according to the queue manager from which the channel is connecting.

Before you begin

Ensure that channel authentication records are enabled as follows:

```
ALTER QMGR CHLAUTH(ENABLED)
```

About this task

Optionally, you can restrict the IP addresses to which the rule applies.

Note that this technique does not apply to server-connection channels. If you specify the name of a server-connection channel in the following commands, it has no effect.

Procedure

- Set a channel authentication record using the MQSC command **`SET CHLAUTH`**, or the PCF command **`Set Channel Authentication Record`**. For example, you can issue the MQSC command:

  ```
  SET CHLAUTH(' generic-channel-name ') TYPE (QMGRMAP) QMNAME(generic-partner-qmgr-name
  ) USERSRC(MAP) MCAUSER(user)
  ```

  generic-channel-name is either the name of a channel to which you want to control access, or a pattern including the asterisk (*) symbol as a wildcard that matches the channel name.
  generic-partner-qmgr-name is either the name of the queue manager, or a pattern including the asterisk (*) symbol as a wildcard that matches the queue manager name.
  user is the user ID to be used for all connections from the specified queue manager.
- To restrict this command to certain IP addresses, include the **ADDRESS** parameter, as follows:

  ```
  SET CHLAUTH(' generic-channel-name ') TYPE (QMGRMAP) QMNAME(generic-partner-qmgr-name
  ) USERSRC(MAP) MCAUSER(user) ADDRESS(
  generic-ip-address)
  ```

  generic-channel-name is either the name of a channel to which you want to control access, or a pattern including the asterisk (*) symbol as a wildcard that matches the channel name.
  generic-ip-address is either a single address, or a pattern including the asterisk (*) symbol as a wildcard or the hyphen (-) to indicate a range, that matches the address. For more information about generic IP addresses, see Generic IP addresses.

Related reference
SET CHLAUTH

Mapping a client user ID to an MCAUSER user ID
You can use a channel authentication record to change the MCAUSER attribute of a server-connection channel, according to the user ID received from a client.

Before you begin

Ensure that channel authentication records are enabled as follows:

```
ALTER QMGR CHLAUTH(ENABLED)
```

About this task

Note that this technique applies only to server-connection channels. It has no effect on other channel types.

Procedure

Set a channel authentication record using the MQSC command **SET CHLAUTH**, or the PCF command **Set Channel Authentication Record** . For example, you can issue the MQSC command:

```
SET CHLAUTH(' generic-channel-name ') TYPE (USERMAP) CLNTUSER(client-user-name) USERSRC(MAP)
MCAUSER(
user)
```

generic-channel-name is either the name of a channel to which you want to control access, or a pattern including the asterisk (*) symbol as a wildcard that matches the channel name.

client-user-name is the user ID associated with the clients connection, the value could be asserted by the client application, altered by connection authentication using early adopt or set via a channel exit.

user is the user ID to be used instead of the client user name.

Related reference

SET CHLAUTH

Attributes of the channels stanza (ChlauthEarlyAdopt)

Mapping an SSL or TLS Distinguished Name to an MCAUSER user ID

You can use a channel authentication record to set the MCAUSER attribute of a channel, according to the Distinguished Name (DN) received.

Before you begin

Ensure that channel authentication records are enabled as follows:

```
ALTER QMGR CHLAUTH(ENABLED)
```

Procedure

Set a channel authentication record using the MQSC command **SET CHLAUTH**, or the PCF command **Set Channel Authentication Record**. For example, you can issue the MQSC command:

```
SET CHLAUTH('generic-channel-name') TYPE (SSLPEERMAP)
SSLPEER(generic-ssl-peer-name) SSLCERTI(generic-issuer-name)
USERSRC(MAP) MCAUSER(user)
```

generic-channel-name is either the name of a channel to which you want to control access, or a pattern including the asterisk (*) symbol as a wildcard that matches the channel name.

generic-ssl-peer-name is a string following the standard IBM MQ rules for SSLPEER values. See IBM MQ rules for SSLPEER values.

user is the user ID to be used for all connections using the specified DN.

generic-issuer-name refers to the Issuer DN of the certificate to match. This parameter is optional but you should use it, to avoid spuriously matching the wrong certificate, if multiple certificate authorities are in use.

Related reference

SET CHLAUTH

Blocking access from a remote queue manager
You can use a channel authentication record to prevent a remote queue manager from starting channels.

Before you begin

Ensure that channel authentication records are enabled as follows:

```
ALTER QMGR CHLAUTH(ENABLED)
```

About this task

Note that this technique does not apply to server-connection channels. If you specify the name of a server-connection channel in the following command, it has no effect.

Procedure

Set a channel authentication record using the MQSC command **`SET CHLAUTH`**, or the PCF command **`Set Channel Authentication Record`**. For example, you can issue the MQSC command:

```
SET CHLAUTH(' generic-channel-name ') TYPE(QMGRMAP) QMNAME(' generic-partner-qmgr-name ')
USERSRC(NOACCESS)
```

generic-channel-name is either the name of a channel to which you want to control access, or a pattern including the asterisk (*) symbol as a wildcard that matches the channel name.
generic-partner-qmgr-name is either the name of the queue manager, or a pattern including the asterisk (*) symbol as a wildcard that matches the queue manager name.

Related reference

SET CHLAUTH

Blocking access for a client user ID
You can use a channel authentication record to prevent a client user ID from establishing a channel connection.

Before you begin

Ensure that channel authentication records are enabled as follows:

```
ALTER QMGR CHLAUTH(ENABLED)
```

About this task

Note that this technique applies only to server-connection channels. It has no effect on other channel types.

Procedure

Set a channel authentication record using the MQSC command **`SET CHLAUTH`**, or the PCF command **`Set Channel Authentication Record`**. For example, you can issue the MQSC command:

```
SET CHLAUTH(' generic-channel-name ') TYPE(USERMAP) CLNTUSER(' client-user-name ')
USERSRC(NOACCESS)
```

generic-channel-name is either the name of a channel to which you want to control access, or a pattern including the asterisk (*) symbol as a wildcard that matches the channel name.
client-user-name is the user ID associated with the clients connection, the value could be asserted by the client application, altered by connection authentication using early adopt or set via a channel exit.

Related reference

SET CHLAUTH

Blocking access for an SSL or TLS Distinguished Name
You can use a channel authentication record to prevent a TLS Distinguished Name (DN) from starting channels.

Before you begin

Ensure that channel authentication records are enabled as follows:

```
ALTER QMGR CHLAUTH(ENABLED)
```

Procedure

Set a channel authentication record using the MQSC command **SET CHLAUTH**, or the PCF command **Set Channel Authentication Record**. For example, you can issue the MQSC command:

```
SET CHLAUTH('generic-channel-name') TYPE(SSLPEERMAP)
SSLPEER('generic-ssl-peer-name') SSLCERTI(generic-issuer-name)
USERSRC(NOACCESS)
```

generic-channel-name is either the name of a channel to which you want to control access, or a pattern including the asterisk (*) symbol as a wildcard that matches the channel name.

generic-ssl-peer-name is a string following the standard IBM MQ rules for SSLPEER values. See IBM MQ rules for SSLPEER values.

generic-issuer-name refers to the Issuer DN of the certificate to match. This parameter is optional but you should use it, to avoid spuriously matching the wrong certificate, if multiple certificate authorities are in use.

Related reference

SET CHLAUTH

Mapping an IP address to an MCAUSER user ID
You can use a channel authentication record to set the MCAUSER attribute of a channel, according to the IP address from which the connection is received.

Before you begin

Ensure that channel authentication records are enabled as follows:

```
ALTER QMGR CHLAUTH(ENABLED)
```

Procedure

Set a channel authentication record using the MQSC command **SET CHLAUTH**, or the PCF command **Set Channel Authentication Record**. For example, you can issue the MQSC command:

```
SET CHLAUTH(' generic-channel-name ') TYPE(ADDRESSMAP) ADDRESS(' generic-ip-address ')
USERSRC(MAP) MCAUSER(user)
```

generic-channel-name is either the name of a channel to which you want to control access, or a pattern including the asterisk (*) symbol as a wildcard that matches the channel name.

user is the user ID to be used for all connections using the specified DN.

generic-ip-address is either the address from which the connection is being made, or a pattern including the asterisk (*) as a wildcard or the hyphen (-) to indicate a range, that matches the address.

Related reference

SET CHLAUTH

Disabling remote access to the queue manager

If you do not want client applications to connect to your queue manager, disable remote access to it.

About this task

Prevent client applications connecting to the queue manager in one of the following ways:

Procedure

- Delete all server-connection channels using the MQSC command **DELETE CHANNEL**.
- Set the message channel agent user identifier (MCAUSER) of the channel to a user ID with no access rights, using the MQSC command **ALTER CHANNEL**.

Setting up connection security

Grant the authority to connect to the queue manager to each user or group of users with a business need to do so.

About this task

To set up connection security, use the appropriate commands for your operating system.

Multi On Multiplatforms, you can also use the SET AUTHREC command.

Procedure

- ALW

  On AIX, Linux, and Windows:

  ```
  setmqaut -m QMgrName -t qmgr -g GroupName +connect
  ```

- IBM i

  On IBM i:

  ```
  GRTMQMAUT OBJ('QMgrName') OBJTYPE(*MQM) USER('GroupName') AUT(*CONNECT)
  ```

- z/OS

  On z/OS:

  ```
  RDEFINE MQCONN QMgrName.BATCH UACC(NONE)
  PERMIT QMgrName.BATCH CLASS(MQCONN) ID(GroupName) ACCESS(READ)
  RDEFINE MQCONN QMgrName.CICS UACC(NONE)
  PERMIT QMgrName.CICS CLASS(MQCONN) ID(GroupName) ACCESS(READ)
  RDEFINE MQCONN QMgrName.IMS UACC(NONE)
  PERMIT QMgrName.IMS CLASS(MQCONN) ID(GroupName) ACCESS(READ)
  RDEFINE MQCONN QMgrName.CHIN UACC(NONE)
  PERMIT QMgrName.CHIN CLASS(MQCONN) ID(GroupName) ACCESS(READ)
  ```

  These commands give authority to connect for batch, CICS, IMS and the channel initiator (CHIN). If you do not use a particular type of connection, omit the relevant commands.

  The variable names have the following meanings:

  QMgrName

  The name of the queue manager. On z/OS, this value can also be the name of a queue sharing group.

ObjectProfile
The name of the object or generic profile for which to change authorizations.

GroupName
The name of the group to be granted access.

Related concepts

"Connection security profiles for the channel initiator" on page 198
Profiles for checking connections from the channel initiator are composed of the queue manager or queue sharing group name followed by the word *CHIN*. Give the user ID used by the channel initiator started task address space READ access to the connection profile.

Controlling user access to queues

You want to control application access to queues. Use this topic to determine what actions to take.

For each true statement in the first column, take the action indicated in the second column.

Statement	Action
The application gets messages from a queue	See "Granting authority to get messages from queues" on page 383
The application sets context	See "Granting authority to set context" on page 384
The application passes context	See "Granting authority to pass context" on page 385
The application puts messages on a clustered queue	See "Authorizing putting messages on remote cluster queues" on page 467
The application puts messages on a local queue	See "Granting authority to put messages to a local queue" on page 386
The application puts messages on a model queue	See "Granting authority to put messages to a model queue" on page 387
The application puts messages on a remote queue	See "Granting authority to put messages to a remote cluster queue" on page 387

Granting authority to get messages from queues
Grant the authority to get messages from a queue or set of queues, to each group of users with a business need for it.

About this task

To grant the authority to get messages from some queues, use the appropriate commands for your operating system.

Multi On Multiplatforms, you can also use the SET AUTHREC command.

Procedure

- Windows
  For AIX, Linux, and Windows systems, issue the following command:

```
setmqaut -m QMgrName -n ObjectProfile -t queue -g GroupName +get
```

- IBM i
  For IBM i, issue the following command:

```
GRTMQMAUT OBJ(' ObjectProfile ') OBJTYPE(*Q) USER(GroupName) AUT(*GET) MQMNAME(' QMgrName ')
```

- z/OS
  For z/OS, issue the following commands:

  ```
  RDEFINE MQQUEUE QMgrName.ObjectProfile UACC(NONE)
  PERMIT QMgrName.ObjectProfile CLASS(MQADMIN) ID(GroupName) ACCESS(UPDATE)
  ```

  The variable names have the following meanings:

  QMgrName
  The name of the queue manager. On z/OS, this value can also be the name of a queue sharing group.

  ObjectProfile
  The name of the object or generic profile for which to change authorizations.

  GroupName
  The name of the group to be granted access.

Granting authority to set context
Grant the authority to set context on a message that is being put, to each group of users with a business need for it.

About this task

To grant the authority to set context on some queues, use the appropriate commands for your operating system.

Multi On Multiplatforms, you can also use the SET AUTHREC command.

Procedure

- ALW
  For AIX, Linux, and Windows systems, issue one of the following commands:
  - To set identity context only:

    ```
    setmqaut -m QMgrName -n ObjectProfile -t queue -g GroupName +setid
    ```

  - To set all context:

    ```
    setmqaut -m QMgrName -n ObjectProfile -t queue -g GroupName +setall
    ```

  Note: To use `setid` or `setall` authority, authorizations must be granted on both the appropriate queue object and also on the queue manager object.

- IBM i
  For IBM i, issue one of the following commands:
  - To set identity context only:

    ```
    GRTMQMAUT OBJ(' ObjectProfile ') OBJTYPE(*Q) USER(GroupName) AUT(*SETID) MQMNAME('
    QMgrName ')
    ```

  - To set all context:

    ```
    GRTMQMAUT OBJ(' ObjectProfile ') OBJTYPE(*Q) USER(GroupName) AUT(*SETALL) MQMNAME('
    QMgrName ')
    ```

- z/OS

  For z/OS, issue one of the following sets of commands:

  - To set identity context only:

    ```
    RDEFINE MQQUEUE QMgrName.ObjectProfile UACC(NONE)
    PERMIT QMgrName.ObjectProfile CLASS(MQQUEUE) ID(GroupName) ACCESS(UPDATE)
    ```

  - To set all context:

    ```
    RDEFINE MQQUEUE QMgrName. ObjectProfile UACC(NONE)
    PERMIT QMgrName.ObjectProfile CLASS(MQQUEUE) ID(GroupName) ACCESS(CONTROL)
    ```

  The variable names have the following meanings:

  QMgrName
  : The name of the queue manager. On z/OS, this value can also be the name of a queue sharing group.

  ObjectProfile
  : The name of the object or generic profile for which to change authorizations.

  GroupName
  : The name of the group to be granted access.

Granting authority to pass context
Grant the authority to pass context from a retrieved message to one that is being put, to each group of users with a business need for it.

About this task

To grant the authority to pass context on some queues, use the appropriate commands for your operating system.

Multi On Multiplatforms, you can also use the SET AUTHREC command.

Procedure

- ALW

  For AIX, Linux, and Windows systems, issue one of the following commands:

  - To pass identity context only:

    ```
    setmqaut -m QMgrName -n ObjectProfile -t queue -g GroupName +passid
    ```

  - To pass all context:

    ```
    setmqaut -m QMgrName -n ObjectProfile -t queue -g GroupName +passall
    ```

- IBM i

  For IBM i, issue one of the following commands:

  - To pass identity context only:

    ```
    GRTMQMAUT OBJ(' ObjectProfile ') OBJTYPE(*Q) USER(GroupName) AUT(*PASSID) MQMNAME('
    QMgrName ')
    ```

  - To pass all context:

```
GRTMQMAUT OBJ(' ObjectProfile ') OBJTYPE(*Q) USER(GroupName) AUT(*PASSALL) MQMNAME('
QMgrName ')
```

- z/OS

  For z/OS, issue the following commands to pass identity context or all context:

  ```
  RDEFINE MQQUEUE QMgrName.ObjectProfile UACC(NONE)
  PERMIT QMgrName.ObjectProfile CLASS(MQQUEUE) ID(GroupName) ACCESS(UPDATE)
  ```

  The variable names have the following meanings:

  QMgrName
  : The name of the queue manager. On z/OS, this value can also be the name of a queue sharing group.

  ObjectProfile
  : The name of the object or generic profile for which to change authorizations.

  GroupName
  : The name of the group to be granted access.

Granting authority to put messages to a local queue
Grant the authority to put messages to a local queue or set of queues, to each group of users with a business need for it.

About this task

To grant the authority to put messages to some local queues, use the appropriate commands for your operating system.

Multi On Multiplatforms, you can also use the SET AUTHREC command.

Procedure

- ALW

  For AIX, Linux, and Windows systems, issue the following command:

  ```
  setmqaut -m QMgrName -n ObjectProfile -t queue -g GroupName +put
  ```

- IBM i

  For IBM i, issue the following command:

  ```
  GRTMQMAUT OBJ(' ObjectProfile ') OBJTYPE(*Q) USER(GroupName) AUT(*PUT) MQMNAME(' QMgrName ')
  ```

- z/OS

  For z/OS, issue the following commands:

  ```
  RDEFINE MQQUEUE QMgrName.ObjectProfile UACC(NONE)
  PERMIT QMgrName.ObjectProfile CLASS(MQQUEUE) ID(GroupName) ACCESS(UPDATE)
  ```

  The variable names have the following meanings:

  QMgrName
  : The name of the queue manager. On z/OS, this value can also be the name of a queue sharing group.

  ObjectProfile
  : The name of the object or generic profile for which to change authorizations.

GroupName
The name of the group to be granted access.

Granting authority to put messages to a model queue
Grant the authority to put messages to a model queue or set of model queues, to each group of users with a business need for it.

About this task

Model queues are used to create dynamic queues. You must therefore grant authority to both the model and dynamic queues. To grant these authorities, use the appropriate commands for your operating system.

Multi On Multiplatforms, you can also use the SET AUTHREC command.

Procedure

- ALW
  For AIX, Linux, and Windows systems, issue the following commands:

```
setmqaut -m QMgrName -n ModelQueueName -t queue -g GroupName +put
setmqaut -m QMgrName -n ObjectProfile -t queue -g GroupName +put
```

- IBM i
  For IBM i, issue the following commands:

```
GRTMQMAUT OBJ(' ModelQueueName ') OBJTYPE(*Q) USER(GroupName) AUT(*PUT) MQMNAME(' QMgrName ')
GRTMQMAUT OBJ(' ObjectProfile ') OBJTYPE(*Q) USER(GroupName) AUT(*PUT) MQMNAME(' QMgrName ')
```

- z/OS
  For z/OS, issue the following commands:

```
RDEFINE MQQUEUE QMgrName.ModelQueueName UACC(NONE)
PERMIT QMgrName.ModelQueueName CLASS(MQQUEUE) ID(GroupName) ACCESS(UPDATE)
RDEFINE MQQUEUE QMgrName.ObjectProfile UACC(NONE)
PERMIT QMgrName.ObjectProfile CLASS(MQQUEUE) ID(GroupName) ACCESS(UPDATE)
```

The variable names have the following meanings:

QMgrName
The name of the queue manager. On z/OS, this value can also be the name of a queue sharing group.

ModelQueueName
The name of the model queue on which dynamic queues are based.

ObjectProfile
The name of the dynamic queue or generic profile for which to change authorizations.

GroupName
The name of the group to be granted access.

Granting authority to put messages to a remote cluster queue
Grant the authority to put messages to a remote cluster queue or set of queues, to each group of users with a business need for it.

About this task

To put a message on a remote cluster queue, you can either put it on a local definition of a remote queue, or a fully qualified remote queue. If you are using a local definition of a remote queue, you need authority to put to the local object: see "Granting authority to put messages to a local queue" on page 386. If you are using a fully qualified remote queue, you need authority to put to the remote queue. Grant this authority using the appropriate commands for your operating system.

The default behavior is to perform access control against the `SYSTEM.CLUSTER.TRANSMIT.QUEUE`. Note that this behavior applies, even if you are using multiple transmission queues.

The specific behavior described in this topic applies only when you have configured the **`ClusterQueueAccessControl`** attribute in the `qm.ini` file to be *RQMName*, as described in the Security stanza topic, and restarted the queue manager.

Multi On Multiplatforms, you can also use the SET AUTHREC command.

Procedure

- ALW
  For AIX, Linux, and Windows systems, issue the following command:

  ```
  setmqaut -m QMgrName -t rqmname -n
  ObjectProfile -g GroupName +put
  ```

  Note that you can use the *rqmname* object for remote cluster queues only.

- IBM i
  For IBM i, issue the following command:

  ```
  GRTMQMAUT OBJTYPE(*RMTMQMNAME) OBJ('
  ObjectProfile') USER(GroupName) AUT(*PUT) MQMNAME('
  QMgrName')
  ```

  Note that you can use the RMTMQMNAME object for remote cluster queues only.

- z/OS
  For z/OS, issue the following commands:

  ```
  RDEFINE MQQUEUE QMgrName.ObjectProfile UACC(NONE)
  PERMIT QMgrName.ObjectProfile CLASS(MQQUEUE) ID(GroupName) ACCESS(UPDATE)
  ```

  Note that you can use the name of the remote queue manager (or queue sharing group) for remote cluster queues only.

  The variable names have the following meanings:

  QMgrName
  The name of the queue manager. On z/OS, this value can also be the name of a queue sharing group.

  ObjectProfile
  The name of the remote queue manager or generic profile for which to change authorizations.

  GroupName
  The name of the group to be granted access.

Controlling user access to topics

You need to control the access of applications to topics. Use this topic to determine what actions to take.

For each true statement in the first column, take the action indicated in the second column.

Table 74. Controlling user access to topics

Statement	Action
The application publishes messages to a topic	See "Granting authority to publish messages to a topic" on page 389

Table 74. Controlling user access to topics (continued)

Statement	Action
The application subscribes to a topic	See "Granting authority to subscribe to topics" on page 389

Granting authority to publish messages to a topic
Grant the authority to publish messages to a topic or set of topics, to each group of users with a business need for it.

About this task

To grant the authority to publish messages to some topics, use the appropriate commands for your operating system.

Multi On Multiplatforms, you can also use the SET AUTHREC command.

Procedure

- ALW
  For AIX, Linux, and Windows systems, issue the following command:

  ```
  setmqaut -m QMgrName -n ObjectProfile -t topic -g GroupName +pub
  ```

- IBM i
  For IBM i, issue the following command:

  ```
  GRTMQMAUT OBJ(' ObjectProfile ') OBJTYPE(*TOPIC) USER(GroupName) AUT(*PUB) MQMNAME('
  QMgrName ')
  ```

- z/OS
  For z/OS, issue the following commands:

  ```
  RDEFINE MQTOPIC QMgrName.ObjectProfile UACC(NONE)
  PERMIT QMgrName.ObjectProfile CLASS(MQTOPIC) ID(GroupName) ACCESS(UPDATE)
  ```

  The variable names have the following meanings:

  QMgrName
  The name of the queue manager. On z/OS, this value can also be the name of a queue sharing group.

  ObjectProfile
  The name of the object or generic profile for which to change authorizations.

  GroupName
  The name of the group to be granted access.

Granting authority to subscribe to topics
Grant the authority to subscribe to a topic or set of topics, to each group of users with a business need for it.

About this task

To grant the authority to subscribe to some topics, use the appropriate commands for your operating system.

Multi On Multiplatforms, you can also use the SET AUTHREC command.

Procedure

- ALW

  For AIX, Linux, and Windows systems, issue the following command:

  ```
  setmqaut -m QMgrName -n ObjectProfile -t topic -g GroupName +sub
  ```

- IBM i

  For IBM i, issue the following command:

  ```
  GRTMQMAUT OBJ(' ObjectProfile ') OBJTYPE(*TOPIC) USER(GroupName) AUT(*SUB) MQMNAME('
  QMgrName ')
  ```

- z/OS

  For z/OS, issue the following commands:

  ```
  RDEFINE MQTOPIC QMgrName.SUBSCRIBE.ObjectProfile UACC(NONE)
  PERMIT QMgrName.SUBSCRIBE.ObjectProfile CLASS(MQTOPIC) ID(GroupName) ACCESS(UPDATE)
  ```

  The variable names have the following meanings:

  QMgrName
  The name of the queue manager. On z/OS, this value can also be the name of a queue sharing group.

  ObjectProfile
  The name of the object or generic profile for which to change authorizations.

  GroupName
  The name of the group to be granted access.

Granting authority to inquire on a queue manager

Grant the authority to inquire on a queue manager, to each group of users with a business need for it.

About this task

To grant the authority to inquire on a queue manager, use the appropriate commands for your operating system.

Multi On Multiplatforms, you can also use the SET AUTHREC command.

Procedure

- ALW

  For AIX, Linux, and Windows systems, issue the following command:

  ```
  setmqaut -m QMgrName -n ObjectProfile -t qmgr -g GroupName +inq
  ```

- IBM i

  For IBM i, issue the following command:

  ```
  GRTMQMAUT OBJ(' ObjectProfile ') OBJTYPE(*MQM) USER(GroupName) AUT(*INQ) MQMNAME(' QMgrName
  ')
  ```

- z/OS

  For z/OS, issue the following commands:

```
RDEFINE MQCMDS QMgrName.ObjectProfile UACC(NONE)
PERMIT QMgrName.ObjectProfile CLASS(MQCMDS) ID(GroupName) ACCESS(READ)
```

These commands grant access to the specified queue manager. To permit the user to use the MQINQ command, issue the following commands:

```
RDEFINE MQCMDS QMgrName.MQINQ.QMGR UACC(NONE)
PERMIT QMgrName.MQINQ.QMGR CLASS(MQCMDS) ID(GroupName) ACCESS(READ)
```

The variable names have the following meanings:

QMgrName
: The name of the queue manager. On z/OS, this value can also be the name of a queue sharing group.

ObjectProfile
: The name of the object or generic profile for which to change authorizations.

GroupName
: The name of the group to be granted access.

Granting authority to access processes

Grant the authority to access a process or set of processes, to each group of users with a business need for it.

About this task

To grant the authority to access some processes, use the appropriate commands for your operating system.

Multi On Multiplatforms, you can also use the SET AUTHREC command.

Procedure

- ALW
  For AIX, Linux, and Windows systems, issue the following command:

  ```
  setmqaut -m QMgrName -n ObjectProfile -t process -g GroupName +all
  ```

- IBM i
  For IBM i, issue the following command:

  ```
  GRTMQMAUT OBJ(' ObjectProfile ') OBJTYPE(*PRC) USER(GroupName) AUT(*ALL) MQMNAME(' QMgrName
  ')
  ```

- z/OS
  For z/OS, issue the following commands:

  ```
  RDEFINE MQPROC QMgrName.ObjectProfile UACC(NONE)
  PERMIT QMgrName.ObjectProfile CLASS(MQPROC) ID(GroupName) ACCESS(READ)
  ```

  The variable names have the following meanings:

  QMgrName
  : The name of the queue manager. On z/OS, this value can also be the name of a queue sharing group.

  ObjectProfile
  : The name of the object or generic profile for which to change authorizations.

GroupName
The name of the group to be granted access.

Granting authority to access namelists

Grant the authority to access a namelist or set of namelists, to each group of users with a business need for it.

About this task

To grant the authority to access some namelists, use the appropriate commands for your operating system.

Multi On Multiplatforms, you can also use the SET AUTHREC command.

Procedure

- ALW
  For AIX, Linux, and Windows systems, issue the following command:

  ```
  setmqaut -m QMgrName -n
  ObjectProfile -t namelist -g GroupName
  +all
  ```

- IBM i
  For IBM i, issue the following command:

  ```
  GRTMQMAUT OBJ('ObjectProfile
  ') OBJTYPE(*NMLIST) USER(GroupName) AUT(*ALL) MQMNAME('
  QMgrName')
  ```

- z/OS
  For z/OS, issue the following commands:

  ```
  RDEFINE MQNLIST
  QMgrName.ObjectProfile UACC(NONE)
  PERMIT QMgrName.ObjectProfile
  CLASS(MQNLIST) ID(GroupName) ACCESS(READ)
  ```

  The variable names have the following meanings:

  QMgrName
  The name of the queue manager. On z/OS, this value can also be the name of a queue sharing group.

  ObjectProfile
  The name of the object or generic profile for which to change authorizations.

  GroupName
  The name of the group to be granted access.

ALW Authority to administer IBM MQ on AIX, Linux, and Windows

IBM MQ administrators can use all IBM MQ commands and grant authorities for other users. When administrators issue commands to remote queue managers, they must have the required authority on the remote queue manager. Further considerations apply to Windows systems.

IBM MQ administrators have authority to use all IBM MQ commands (including the commands to grant IBM MQ authorities for other users).

To be an IBM MQ administrator, you must be a member of a special group that is called the **`mqm`** group.

Windows Alternatively, on Windows only, local accounts can administer IBM MQ if they are members of the Administrators group on Windows systems.

Attention: You can add your Azure AD user to the mqm group by using an administrator command. For example, use the command `net localgroup mqm AzureAD\<your userID> /add`. Then run IBM MQ administration commands or use IBM MQ Explorer.

The **mqm** group is created automatically when IBM MQ is installed. You can add further users to the group to allow them to perform administration. All members of this group have access to all resources. This access can be revoked only by removing a user from the **mqm** group and issuing the **`REFRESH SECURITY`** command.

Administrators can use control commands to administer IBM MQ. One of these control commands is **`setmqaut`**, which is used to grant authorities to other users to enable them to access or control IBM MQ resources. The PCF commands for managing authority records are available to non-administrators who are granted dsp and chg authorities on the queue manager. For more information about managing authorities by using PCF commands, see Programmable Command Formats.

Administrators must have the required authorities for the MQSC commands to be processed by the remote queue manager. The IBM MQ Explorer issues PCF commands to perform administration tasks. Administrators require no additional authorities to use the IBM MQ Explorer to administer a queue manager on the local system. When the IBM MQ Explorer is used to administer a queue manager on another system, administrators must have the required authorities for the PCF commands to be processed by the remote queue manager.

Attention: You do not have to be an administrator to use the control command **`runmqsc`**, that issues IBM MQ Script (MQSC) commands.

When **`runmqsc`** is used in indirect mode to send MQSC commands to a remote queue manager, each MQSC command is encapsulated within an Escape PCF command.

For more information about authority checks when PCF and MQSC commands are processed, see the following topics:

- For PCF commands that operate on queue managers, queues, processes, namelists, and authentication information objects, see Authority to work with IBM MQ objects. Refer to this section for the equivalent MQSC commands encapsulated within Escape PCF commands.
- For PCF commands that operate on channels, channel initiators, listeners, and clusters, see Channel security.
- For PCF commands that operate on authority records, see Authority checking for PCF commands
- z/OS For MQSC commands that are processed by the command server on IBM MQ for z/OS, see Command security and command resource security on z/OS .

Additionally, on Windows systems, the SYSTEM account has full access to IBM MQ resources.

On AIX and Linux platforms, a special user ID of **mqm** is also created, for use by the product only. It must never be available to non-privileged users. All IBM MQ objects are owned by user ID **mqm**.

On Windows systems, members of the Administrators group can also administer any queue manager, as can the SYSTEM account. You can also create a domain **mqm** group on the domain controller that contains all privileged user IDs active within the domain, and add it to the local **mqm** group. Some commands, for example **`crtmqm`**, manipulate authorities on IBM MQ objects and so need authority to work with these objects (as described in the following sections). Members of the **mqm** group have authority to work with all objects, but there might be circumstances on Windows systems when authority is denied if you have a local user and a domain-authenticated user with the same name. This is described in "Principals and groups on AIX, Linux, and Windows" on page 397.

Windows versions with a User Account Control (UAC) feature restricts the actions users can perform on certain operating system facilities, even if they are members of the Administrators group. If your userid is in the Administrators group but not the **mqm** group you must use an elevated command prompt to issue IBM MQ admin commands such as **`crtmqm`**, otherwise the error `AMQ7077: You are not authorized` `to perform the requested operation` is generated. To open an elevated command prompt, right-click the start menu item, or icon, for the command prompt, and select **Run as administrator**.

You do not need to be a member of the **mqm** group to take the following actions:

- Issue commands from an application program that issues PCF commands, or MQSC commands within an Escape PCF command, unless the commands manipulate channel initiators. (These commands are described in "Protecting channel initiator definitions" on page 115).
- Issue MQI calls from an application program (unless you want to use the fast path bindings on the `MQCONNX` call).
- Use the `crtmqcvx` command to create a fragment of code that performs data conversion on data type structures.
- Use the `dspmq` command to display queue managers.
- Use the `dspmqtrc` command to display IBM MQ formatted trace output.

A 12 character limitation applies to both group and user IDs.

UNIX and Linux platforms generally restrict the length of a user ID to 12 characters. AIX 5.3 has raised this limit but IBM MQ continues to observe a 12 character restriction on all UNIX and Linux platforms. If you use a user ID of greater than 12 characters, IBM MQ replaces it with the value `UNKNOWN`. Do not define a user ID with a value of `UNKNOWN`.

ALW Managing the mqm group on AIX, Linux, and Windows

Users in the mqm group are granted full administrative privileges over IBM MQ. For this reason, you should not enroll applications and ordinary users in the mqm group. The mqm group should contain the accounts of the IBM MQ administrators only.

These tasks are described in:

- Windows Creating and managing groups on Windows
- AIX Creating and managing groups on AIX
- Linux Creating and managing groups on Linux

Windows If your domain controller runs on Windows 2000 or Windows 2003 or later, your domain administrator might have to set up a special account for IBM MQ to use. For more information, see Configuring IBM MQ with the Prepare IBM MQ Wizard and Creating and setting up Windows domain accounts for IBM MQ.

ALW Authority to work with IBM MQ objects on AIX, Linux, and Windows

All objects are protected by IBM MQ, and principals must be given appropriate authority to access them. Different principals need different access rights to different objects.

Queue managers, queues, process definitions, namelists, channels, client connection channels, listeners, services, and authentication information objects are all accessed from applications that use MQI calls or PCF commands. These resources are all protected by IBM MQ, and applications need to be given permission to access them. The entity making the request might be a user, an application program that issues an MQI call, or an administration program that issues a PCF command. The identifier of the requester is referred to as the *principal*.

Different groups of principals can be granted different types of access authority to the same object. For example, for a specific queue, one group might be allowed to perform both put and get operations; another group might be allowed only to browse the queue (`MQGET` with browse option). Similarly, some groups might have put and get authority to a queue, but not be allowed to alter attributes of the queue or delete it.

Some operations are particularly sensitive and should be limited to privileged users. For example:

- Accessing some special queues, such as transmission queues or the command queue SYSTEM.ADMIN.COMMAND.QUEUE
- Running programs that use full MQI context options
- Creating and deleting application queues

Full access permission to an object is automatically given to the user ID that created the object and to all members of the mqm group (and to the members of the local Administrators group on Windows systems).

Related concepts

"Authority to administer IBM MQ on AIX, Linux, and Windows" on page 392
IBM MQ administrators can use all IBM MQ commands and grant authorities for other users. When administrators issue commands to remote queue managers, they must have the required authority on the remote queue manager. Further considerations apply to Windows systems.

ALW When security checks are made on AIX, Linux, and Windows

Security checks are typically made on connecting to a queue manager, opening or closing objects, and putting or getting messages.

The security checks made for a typical application are as follows:

Connecting to the queue manager (MQCONN or MQCONNX calls)

This is the first time that the application is associated with a particular queue manager. The queue manager interrogates the operating environment to discover the user ID associated with the application. IBM MQ then verifies that the user ID is authorized to connect to the queue manager and retains the user ID for future checks.

Users do not have to sign on to IBM MQ; IBM MQ assumes that users have signed on to the underlying operating system and have been authenticated by that.

Opening the object (MQOPEN or MQPUT1 calls)

IBM MQ objects are accessed by opening the object and issuing commands against it. All resource checks are performed when the object is opened, rather than when it is actually accessed. This means that the **MQOPEN** request must specify the type of access required (for example, whether the user wants only to browse the object or perform an update like putting messages onto a queue).

IBM MQ checks the resource that is named in the **MQOPEN** request. For an alias or remote queue object, the authorization used is that of the object itself, not the queue to which the alias or remote queue resolves. This means that the user does not need permission to access it. Limit the authority to create queues to privileged users. If you do not, users might bypass the normal access control simply by creating an alias. If a remote queue is referred to explicitly with both the queue and queue manager names, the transmission queue associated with the remote queue manager is checked.

The authority to a dynamic queue is based on that of the model queue from which it is derived, but is not necessarily the same. This is described in Note "1" on page 133.

The user ID used by the queue manager for access checks is the user ID obtained from the operating environment of the application connected to the queue manager. A suitably authorized application can issue an **MQOPEN** call specifying an alternative user ID; access control checks are then made on the alternative user ID. This does not change the user ID associated with the application, only that used for access control checks.

Putting and getting messages (MQPUT or MQGET calls)

No access control checks are performed.

Closing the object (MQCLOSE)

No access control checks are performed, unless the **MQCLOSE** results in a dynamic queue being deleted. In this case, there is a check that the user ID is authorized to delete the queue.

Subscribing to a topic (MQSUB)

When an application subscribes to a topic, it specifies the type of operation that it needs to perform. It is either creating a new subscription, altering an existing subscription, or resuming an existing subscription without changing it. For each type of operation, the queue manager checks that the user ID that is associated with the application has the authority to perform the operation.

When an application subscribes to a topic, the authority checks are performed against the topic objects that are found in the topic tree at or above the point in the topic tree at which the application subscribed. The authority checks might involve checks on more than one topic object.

The user ID that the queue manager uses for the authority checks is the user ID obtained from the operating system when the application connects to the queue manager.

The queue manager performs authority checks on subscriber queues but not on managed queues.

ALW How access control is implemented by IBM MQ on AIX, Linux, and Windows

IBM MQ uses the security services provided by the underlying operating system, using the object authority manager. IBM MQ supplies commands to create and maintain access control lists.

An access control interface called the Authorization Service Interface is part of IBM MQ. IBM MQ supplies an implementation of an access control manager (conforming to the Authorization Service Interface) known as the *object authority manager (OAM)*. This is automatically installed and enabled for each queue manager you create, unless you specify otherwise (as described in "Preventing security access checks on AIX, Linux, and Windows systems" on page 355). The OAM can be replaced by any user or vendor written component that conforms to the Authorization Service Interface.

The OAM exploits the security features of the underlying operating system, using operating system user and group IDs. Users can access IBM MQ objects only if they have the correct authority. "Controlling access to objects by using the OAM on AIX, Linux, and Windows" on page 345 describes how to grant and revoke this authority.

The OAM maintains an access control list (ACL) for each resource that it controls. Authorization data is stored on a local queue called SYSTEM.AUTH.DATA.QUEUE. Access to this queue is restricted to users in the mqm group, and additionally on Windows, to users in the Administrators group, and users logged in with the SYSTEM ID. User access to the queue cannot be changed.

IBM MQ supplies commands to create and maintain access control lists. For more information on these commands, see "Controlling access to objects by using the OAM on AIX, Linux, and Windows" on page 345.

IBM MQ passes the OAM a request containing a principal, a resource name, and an access type. The OAM grants or rejects access based on the ACL that it maintains. IBM MQ follows the decision of the OAM; if the OAM cannot make a decision, IBM MQ does not allow access.

ALW Identifying the user ID on AIX, Linux, and Windows

The object authority manager identifies the principal that is requesting access to a resource. The user ID used as the principal varies according to context.

The object authority manager (OAM) must be able to identify who is requesting access to a particular resource. IBM MQ uses the term *principal* to refer to this identifier. The principal is established when the application first connects to the queue manager; it is determined by the queue manager from the user ID associated with the connecting application. (If the application issues XA calls without connecting to the queue manager, then the user ID associated with the application that issues the xa_open call is used for authority checks by the queue manager.)

On AIX and Linux systems, the authorization routines checks either the real (logged-in) user ID, or the effective user ID associated with the application. The user ID checked can be dependent on the bind type, for details see Installable services.

IBM MQ propagates the user ID received from the system in the message header (MQMD structure) of each message as identification of the user. This identifier is part of the message context information and is described in "Context authority on AIX, Linux, and Windows" on page 399. Applications cannot alter this information unless they have been authorized to change context information.

ALW *Principals and groups on AIX, Linux, and Windows*

Principals can belong to groups. By granting resource access to groups rather than to individuals, you can reduce the amount of administration required. Access Control Lists (ACLs) are based on both groups and user IDs.

For example, you might define a group consisting of users who want to run a particular application. Other users can be given access to all the resources they require by adding their user ID to the appropriate group.

This process of defining and managing groups is described for particular platforms:

- AIX Creating and managing groups on AIX
- Linux Creating and managing groups on Linux
- Windows Creating and managing groups on Windows

A principal can belong to more than one group (its group set). It has the aggregate of all the authorities granted to each group in its group set. These authorities are cached, so any changes you make to the group membership of the principal are not recognized until the queue manager is restarted, unless you issue the MQSC command **`REFRESH SECURITY`** (or its PCF equivalent).

Linux AIX AIX and Linux systems

Access control lists (ACLs) are based on both user IDs and groups and you can use either for authorization by setting the **`SecurityPolicy`** attribute to the appropriate value as described in Service stanza of the qm.ini file.

You can use the *user-based model* for authorization, and this allows you to use both users and groups. However, when you specify a user in the setmqaut command, the new permissions apply to that user alone, and not any groups to which that user belongs. For more information, see "OAM user-based permissions on AIX and Linux" on page 345.

When you use the *group-based model* for authorization, the primary group to which the user ID belongs is included in the ACL. The individual user ID is not included and authority is granted to all members of that group. Because of this, be aware that you can inadvertently change the authority of a principal by changing the authority of another principal in the same group.

All users are nominally assigned to the default user group nobody and by default, no authorizations are given to this group. You can change the authorization in the nobody group to grant access to IBM MQ resources to users without specific authorizations.

From IBM MQ 9.3.0, you can use the `UserExternal` option of the **`SecurityPolicy`** attribute to create a non-operating system user name. If you create a non-operating system user name, that user is considered to belong to no groups, except the nobody group. For more information about this option, see crtmqm and Service stanza of the qm.ini file.

Do not define a user ID with the value `UNKNOWN`. The value `UNKNOWN` is used when a user ID is too long, so arbitrary user IDs would use the access authorities of `UNKNOWN`.

See "Setting authorizations" on page 405 for information on using LDAP.

User IDs can contain up to 12 characters and group names up to 12 characters.

Windows Windows systems

ACLs are based on both user IDs and groups. Checks are the same as for AIX and Linux. You can have different users on different domains with the same user ID. IBM MQ permits user IDs to be qualified by a domain name so that these users can be given different levels of access.

The group name can optionally include a domain name, specified in the following formats:

```
GroupName@domain domain_name\group_name
```

Global groups are checked by the OAM in two cases only:

1. The queue manager security stanza includes the setting: `GroupModel=GlobalGroups`. See Securing.
2. The queue manager is using an alternative security access group. See **crtmqm** .

User IDs can contain up to 20 characters, domain names up to 15 characters, and group names up to 64 characters.

The OAM first checks the local security database, then the database of the primary domain, and finally the database of any trusted domains. The first user ID encountered is used by the OAM for checking. Each of these user IDs might have different group memberships on a particular computer.

Some control commands (for example, **crtmqm**) change authorities on IBM MQ objects using the object authority manager (OAM). The OAM searches the security databases in the order given in the preceding paragraph to determine the authority rights for a particular user ID. As a result, the authority determined by the OAM might override the fact that a user ID is a member of the local mqm group. For example, if you issue the **crtmqm** command from a user ID authenticated by a domain controller that has membership of the local mqm group through a global group, the command fails if the system has a local user with the same name who is not in the local mqm group.

For more information about setting the **SecurityPolicy** attribute on Windows, see Service stanza of the qm.ini file.

Windows *Windows security identifiers (SIDs)*

IBM MQ on Windows uses the SID where it is available. If a Windows SID is not supplied with an authorization request, IBM MQ identifies the user based on the user name alone, but this might result in the wrong authority being granted.

On Windows systems, the security identifier (SID) is used to supplement the user ID. The SID contains information that identifies the full user account details on the Windows security account manager (SAM) database where the user is defined. When a message is created on IBM MQ for Windows, IBM MQ stores the SID in the message descriptor. When IBM MQ on Windows performs authorization checks, it uses the SID to query the full information from the SAM database. (The SAM database in which the user is defined must be accessible for this query to succeed.)

By default, if a Windows SID is not supplied with an authorization request, IBM MQ identifies the user based on the user name alone. It does this by searching the security databases in the following order:

1. The local security database
2. The security database of the primary domain
3. The security database of trusted domains

If the user name is not unique, incorrect IBM MQ authority might be granted. To prevent this problem, include an SID in each authorization request; the SID is used by IBM MQ to establish user credentials.

To specify that all authorization requests must include an SID, use **regedit**. Set the SecurityPolicy to NTSIDsRequired.

ALW Alternate-user authority on AIX, Linux, and Windows

You can specify that a user ID can use the authority of another user when accessing an IBM MQ object. This is called *alternate-user authority*, and you can use it on any IBM MQ object.

Alternate-user authority is essential where a server receives requests from a program and wants to ensure that the program has the required authority for the request. The server might have the required authority, but it needs to know whether the program has the authority for the actions it has requested.

For example, assume that a server program running under user ID PAYSERV retrieves a request message from a queue that was put on the queue by user ID USER1. When the server program gets the request message, it processes the request and puts the reply back into the reply-to queue specified with the request message. Instead of using its own user ID (PAYSERV) to authorize opening the reply-to queue, the server can specify a different user ID, in this case, USER1. In this example, you can use alternate-user authority to control whether PAYSERV is allowed to specify USER1 as an alternate-user ID when it opens the reply-to queue.

The alternate-user ID is specified on the **AlternateUserId** field of the object descriptor.

Linux Solving certain group membership problems on Linux

Some systems are slow to return group information through the normal series of **getgrent** operating system API calls and if your enterprise has thousands of groups to search, looking for which groups the mqm user is in, the slow response can cause an internal queue manager timeout. To circumvent this problem there is an alternative operating system API.

To use the alternative API that is faster, and returns all groups from one call, set the environment variable MQS_GETGROUPLIST_API.

You might have received an RC2035 error when granting connect access to the secondary group of the user and enabling the MQS_GETGROUPLIST_API variable alleviates the problem.

IBM MQ then uses the **getgrouplist** API instead of the **getgrent** API.

To enable **getgrouplist**:

1. Stop the queue manager
2. Issue the command export MQS_GETGROUPLIST_API=1
3. Restart the queue manager

Retry the scenario that failed and if your problem has been solved, you might consider modifying the `.bashrc / .profile` file for the user mqm to add this environment variable, or add the environment variable into the script you use to start the queue manager.

If your system merges user or group information for the operating system from multiple repositories such as NIS or LDAP, then ensure the group or user ID is consistent across all repositories including the local one, as these are used to install and set operating system level permissions.

ALW Context authority on AIX, Linux, and Windows

Context is information that applies to a particular message and is contained in the message descriptor, MQMD, which is part of the message. Applications can specify the context data when either an `MQOPEN` or `MQPUT` call is made.

The context information comes in two sections:

Identity section
Who the message came from. It consists of the `UserIdentifier`, `AccountingToken`, and `ApplIdentityData` fields.

Origin section
Where the message came from, and when it was put onto the queue. It consists of the `PutApplType`, `PutApplName`, `PutDate`, `PutTime`, and `ApplOriginData` fields.

Applications can specify the context data when either an `MQOPEN` or `MQPUT` call is made. This data might be generated by the application, passed on from another message, or generated by the queue manager by default. For example, context data can be used by server programs to check the identity of the requester, testing whether the message came from an application running under an authorized user ID.

A server program can use the `UserIdentifier` to determine the user ID of an alternative user. You use context authorization to control whether the user can specify any of the context options on any `MQOPEN` or `MQPUT1` call.

See Controlling context information for information about the context options, and MQMD - Message descriptor for descriptions of the message descriptor fields relating to context.

Implementing access control in security exits

You can implement access control in a security exit by use of the MCAUserIdentifier or the object authority manager.

MCAUserIdentifier

Every instance of a channel that is current has an associated channel definition structure, MQCD. The initial values of the fields in MQCD are determined by the channel definition that is created by an IBM MQ administrator. In particular, the initial value of one of the fields, *MCAUserIdentifier*, is determined by the value of the MCAUSER parameter on the DEFINE CHANNEL command, or by the equivalent to MCAUSER if the channel definition is created in another way.

The MQCD structure is passed to a channel exit program when it is called by an MCA. When a security exit is called by an MCA, the security exit can change the value of *MCAUserIdentifier*, replacing any value that was specified in the channel definition.

Multi On Multiplatforms, unless the value of *MCAUserIdentifier* is blank, the queue manager uses the value of *MCAUserIdentifier* as the user ID for authority checks when an MCA attempts to access the queue manager's resources after it has connected to the queue manager. If the value of *MCAUserIdentifier* is blank, the queue manager uses the default user ID of the MCA instead. This applies to RCVR, RQSTR, CLUSRCVR and SVRCONN channels. For sending MCAs, the default user ID is always used for authority checks, even if the value of *MCAUserIdentifier* is not blank.

z/OS On z/OS, the queue manager might use the value of *MCAUserIdentifier* for authority checks, provided it is not blank. For receiving MCAs and server connection MCAs, whether the queue manager uses the value of *MCAUserIdentifier* for authority checks depends on:

- The value of the PUTAUT parameter in the channel definition
- The RACF profile used for the checks
- The access level of the channel initiator address space user ID to the RESLEVEL profile

For sending MCAs, it depends on:

- Whether the sending MCA is a caller or a responder
- The access level of the channel initiator address space user ID to the RESLEVEL profile

The user ID that a security exit stores in *MCAUserIdentifier* can be acquired in various ways. Here are some examples:

- Provided there is no security exit at the client end of an MQI channel, a user ID associated with the IBM MQ client application flows from the client connection MCA to the server connection MCA when the client application issues an MQCONN call. The server connection MCA stores this user ID in the *RemoteUserIdentifier* field in the channel definition structure, MQCD. If the value of *MCAUserIdentifier* is blank at this time, the MCA stores the same user ID in *MCAUserIdentifier*. If the MCA does not store the user ID in *MCAUserIdentifier*, a security exit can do it later by setting *MCAUserIdentifier* to the value of *RemoteUserIdentifier*.

  If the user ID that flows from the client system is entering a new security domain and is not valid on the server system, the security exit can substitute the user ID for one that is valid and store the substituted user ID in *MCAUserIdentifier*.
- The user ID can be sent by the partner security exit in a security message.

  On a message channel, a security exit called by the sending MCA can send the user ID under which the sending MCA is running. A security exit called by the receiving MCA can then store the user ID in *MCAUserIdentifier*. Similarly, on an MQI channel, a security exit at the client end of the channel can send the user ID associated with the IBM MQ MQI client application. A security exit at the server end of the channel can then store the user ID in *MCAUserIdentifier*. As in the previous example, if the user ID is not valid on the target system, the security exit can substitute the user ID for one that is valid and store the substituted user ID in *MCAUserIdentifier*.

If a digital certificate is received as part of the identification and authentication service, a security exit can map the Distinguished Name in the certificate to a user ID that is valid on the target system. It can then store the user ID in *MCAUserIdentifier*.

- If TLS is used on the channel, the partner's Distinguished Name (DN) is passed to the exit in the SSLPeerNamePtr field of the MQCD, and the DN of the issuer of that certificate is passed to the exit in the SSLRemCertIssNamePtr field of the MQCXP.

For more information about the *MCAUserIdentifier* field, the channel definition structure, MQCD, and the channel exit parameter structure, MQCXP, see Channel-exit calls and data structures. For more information about the user ID that flows from a client system on an MQI channel, see Access control.

Note: Security exit applications constructed prior to the release of IBM WebSphere MQ 7.1 might require updating. For more information see Channel security exit programs.

IBM MQ object authority manager user authentication

On IBM MQ MQI client connections, security exits can be used to modify or create the MQCSP structure used in object authority manager (OAM) user authentication. This is described in Channel-exit programs for messaging channels

Implementing access control in message exits

You might need to use a message exit to substitute one user ID with another.

Consider a client application that sends a message to a server application. The server application can extract the user ID from the *UserIdentifier* field in the message descriptor and, provided it has alternate user authority, ask the queue manager to use this user ID for authority checks when it accesses IBM MQ resources on behalf of the client.

If the PUTAUT parameter is set to CTX (or ALTMCA on z/OS) in the channel definition, the user ID in the *UserIdentifier* field of each incoming message is used for authority checks when the MCA opens the destination queue.

In certain circumstances, when a report message is generated, it is put using the authority of the user ID in the *UserIdentifier* field of the message causing the report. In particular, confirm-on-delivery (COD) reports and expiration reports are always put with this authority.

Because of these situations, it might be necessary to substitute one user ID for another in the *UserIdentifier* field as a message enters a new security domain. This can be done by a message exit at the receiving end of the channel. Alternatively, you can ensure that the user ID in the *UserIdentifier* field of an incoming message is defined in the new security domain.

If an incoming message contains a digital certificate for the user of the application that sent the message, a message exit can validate the certificate and map the Distinguished Name in the certificate to a user ID that is valid on the receiving system. It can then set the *UserIdentifier* field in the message descriptor to this user ID.

If it is necessary for a message exit to change the value of the *UserIdentifier* field in an incoming message, it might be appropriate for the message exit to authenticate the sender of the message at the same time. For more details, see “Identity mapping in message exits” on page 317.

Implementing access control in the API exit and API-crossing exit

An API or API-crossing exit can provide access controls to supplement those provided by IBM MQ. In particular, the exit can provide access control at the message level. The exit can ensure that an application puts on a queue, or gets from a queue, only those messages that satisfy certain criteria.

Consider the following examples:

- A message contains information about an order. When an application attempts to put a message on a queue, an API or API-crossing exit can check that the total value of the order is less than some prescribed limit.

- Messages arrive on a destination queue from remote queue managers. When an application attempts to get a message from the queue, an API or API-crossing exit can check that the sender of the message is authorized to send a message to the queue.

Multi Streaming queues security on Multiplatforms

The streaming queues feature allows an administrator to configure a local (or model) queue with a secondary queue, where duplicate messages are placed, whenever a message is put to the original queue. There are two aspects to consider regarding queue streaming authorities.

Authority to configure a queue for streaming duplicate messages

If you want to enable message streaming of duplicate messages from one queue to a secondary queue, you must have permission to do so. Permission to configure the **STREAMQ** attribute of a queue requires that you have the following authorities:

1. CHG authority of the queue they are altering the **STREAMQ** attribute for
2. CHG authority of the queue you want duplication messages to be put to

The combination of these two authority checks at configuration time ensures that a user, who only has CHG authority on the original queue, cannot cause messages to be put to another queue on which they have no permissions.

Authority to open the queue or queues and put messages

When an application opens a queue that has been configured with a secondary queue, through its **STREAMQ** attribute, an authority check is made that the application user has PUT authority on the original queue.

Note: No additional authority check is made for the application user on the secondary queue, which is similar to the authority model used for alias queues.

Applications consuming messages from either the original or the secondary queue require GET or BROWSE authority, only on the queue they are consuming from.

No additional authority checks are made at put or get time.

Example

The following example shows the correct authorities being set to allow user `admin` to configure an original queue, INQUIRIES.QUEUE, to stream its duplicate messages to local queue ANALYTICS.QUEUE, but preventing `admin` from duplicating messages to PURCHASES.QUEUE:

```
SET AUTHREC PROFILE(INQUIRIES.QUEUE) PRINCIPAL('admin') AUTHADD(CHG)
SET AUTHREC PROFILE(ANALYTICS.QUEUE) PRINCIPAL('admin') AUTHADD(CHG)
SET AUTHREC PROFILE(PURCHASES.QUEUE) PRINCIPAL('admin') AUTHRMV(CHG)
```

User `admin` is then able to issue the following command:

```
ALTER QLOCAL(INQUIRIES.QUEUE) STREAMQ(ANALYTICS.QUEUE)
```

but if the same user issues the following command:

```
ALTER QLOCAL(INQUIRIES.QUEUE) STREAMQ(PURCHASES.QUEUE)
```

to configure INQUIRIES.QUEUE to put duplicate messages to PURCHASES.QUEUE, they receive the following error:

```
AMQ8135E  Not Authorized
```

With INQUIRIES.QUEUE configured to duplicate messages to ANALYTICS.QUEUE, the following authority records are used to allow an application running as user `appuser` to put messages to INQUIRIES.QUEUE, and duplicate messages to ANALYTICS.QUEUE:

```
SET AUTHREC PROFILE(INQUIRIES.QUEUE) PRINCIPAL('appuser') AUTHADD(PUT)
```

Note: `appuser` does not require an authority record on ANALYTICS.QUEUE. Duplicate messages are put to the queue by the queue manager.

Related concepts
Streaming queues

z/OS Streaming queues security on z/OS

The streaming queues feature allows an administrator to configure a local (or model) queue with a secondary queue, where duplicate messages are placed, whenever a message is put to the original queue. There are two aspects to consider regarding queue streaming authorities.

Authority to configure a queue for streaming duplicate messages

If you want to enable message streaming of duplicate messages from one queue to a secondary queue, you must have permission to do so. Permission to configure the **STREAMQ** attribute of a queue requires that you have the following profiles setup:

1. ALTER access level to MQADMIN or MXADMIN for the queue they are altering the **STREAMQ** attribute for
2. ALTER access level to MQADMIN or MXADMIN for the queue you want to stream messages to

The combination of these security checks at configuration time ensures that a user, who only has ALTER access on the original queue, cannot cause messages to be put to another queue on which they have no permissions.

Authority to open the queue or queues and put messages

When an application opens a queue that has been configured with a secondary queue, through its **STREAMQ** attribute, an authority check is made that the application user has UPDATE authority on the original queue.

Note: No additional authority check is made for the application user on the secondary queue, which is similar to the authority model used for alias queues.

Applications consuming messages from either the original or the secondary queue require UPDATE or READ authority, only on the queue they are consuming from.

No additional authority checks are made at put or get time.

Example

The following example shows the correct profiles being set to allow user ADMIN to configure an original queue, INQUIRIES.QUEUE, to stream messages to local queue ANALYTICS.QUEUE using RACF:

```
RDEFINE MQCMDS <QMGR>.ALTER.QLOCAL UACC(NONE) OWNER(<OWNER>)
PERMIT <QMGR>.ALTER.QLOCAL CLASS(MQCMDS) ID(ADMIN) ACCESS(ALTER)

RDEFINE MQADMIN <QMGR>.QUEUE.INQUIRIES.QUEUE UACC(NONE) OWNER(<OWNER>)
PERMIT <QMGR>.QUEUE.INQUIRIES.QUEUE CLASS(MQADMIN) ID(ADMIN) ACCESS(ALTER)

RDEFINE MQADMIN <QMGR>.QUEUE.ANALYTICS.QUEUE UACC(NONE) OWNER(<OWNER>)
PERMIT <QMGR>.QUEUE.ANALYTICS.QUEUE CLASS(MQADMIN) ID(ADMIN) ACCESS(ALTER)
```

User ADMIN is then able to issue the following command:

```
ALTER QLOCAL(INQUIRIES.QUEUE) STREAMQ(ANALYTICS.QUEUE)
```

but if the same user issues the following command without setting up the correct security profiles:

```
ALTER QLOCAL(INQUIRIES.QUEUE) STREAMQ(PURCHASES.QUEUE)
```

to configure INQUIRIES.QUEUE to put duplicate messages to PURCHASES.QUEUE, they receive the following error:

`CSQM166I <QMGR> CSQMAQLC QLOCAL(INQUIRIES.QUEUE) NOT AUTHORIZED`

Related concepts

Streaming queues

Multi LDAP authorization on Multiplatforms

You can use LDAP authorization to remove the need for a local user ID.

Availability of LDAP authorization on supported platforms

LDAP authorization is available on Multiplatforms:

Attention:

From IBM MQ 9.0 general availability, this functionality is available on all queue managers, whether new or migrated from an earlier release.

Overview of LDAP authorization

With LDAP authorization, commands that handle authorization configuration, such as **setmqaut** and **DISPLAY AUTHREC**, can process Distinguished Names. Previously, users were authenticated by comparing their credentials with the maximum available characters that exist for users and groups on the local operating system.

Attention: If you have run the **DEFINE AUTHINFO** command, you must restart the queue manager. If you do not restart the queue manager, the setmqaut command does not return the correct result.

If a user provides a user ID, rather than a Distinguished Name, the user ID is processed. For example, when there is an incoming message on a channel with PUTAUT(CTX), the characters in the user ID are mapped to an LDAP Distinguished Name, and the appropriate authorization checks are made.

Other commands such as **DISPLAY CONN**, continue to work with and show the actual value for the user ID, even though that user ID might not actually exist on the local OS.

Linux AIX When LDAP authorization is in place, the queue manager always uses the user model of security on AIX and Linux platforms, regardless of the **SecurityPolicy** attribute in the `qm.ini` file. So setting permissions for an individual user affects only that user, and not anyone else who belongs to any of that user's groups.

As with the OS model, a user still has the combined authority that has been assigned to both the individual and to all of the groups (if any) to which the user belongs.

For example, assume that the following records have been defined in an LDAP repository.

- In the **inetOrgPerson** class:

```
dn="cn=JohnDoe, ou=users, o=yourcompany, c=yourcountry"
        email=JohnDoe1@yourcompany.com [longer than 12 characters]
        shortu=jodoe
        Phone=1234567
```

- In the **groupOfNames** class:

```
dn="cn=Application Group A, ou=groups, o=yourcompany, c=yourcountry"
        longname=ApplicationGroupA [longer than 12 characters]
        members="cn=JaneDoe, ou=users, o=yourcompany, c=yourcountry",
                "cn=JohnDoe, ou=users, o=yourcompany, c=yourcountry"
```

For authentication purposes, a queue manager using this LDAP server must have been defined so that its **CONNAUTH** value points at an **AUTHINFO** object of type `IDPWLDAP`, and whose relevant name-resolution attributes are probably set as follows:

```
USRFIELD(email) SHORTUSR(shortu)
BASEDNU(ou=users,o=yourcompany,c=yourcountry) CLASSUSR(inetOrgPerson)
```

Given this configuration for authentication, an application can complete the CSPUserID field, used within the MQCNO call, with either of the following sets of values:

```
" cn=JohnDoe ", " JohnDoe1@yourcompany.com ", " email=JohnDoe1@yourcompany.com "
```

or

```
" cn=JohnDoe, ou=users, o=ibm, c=uk ", " shortu=jodoe "
```

In either case, the system can use the supplied values to authenticate the OS context of " `jodoe`".

Multi Setting authorizations

How you use the short name or **USRFIELD** to set authorizations.

The approach of working with multiple formats, described in "LDAP authorization on Multiplatforms" on page 404, continues into the authorization commands, with a further extension that either the `shortname` or the USRFIELD can be used in an unadorned fashion.

The character string specifies a particular attribute in the LDAP record when naming users (principals) for authorization.

Important: The character string must not contain the = character, because this character cannot be used in an operating system user ID.

If you pass a principal name to the OAM for authorization that is potentially a `shortname`, the character string must fit into 12 characters. The mapping algorithm first tries to resolve it to a DN using the SHORTUSR attribute in its LDAP query.

If that fails with an UNKNOWN_ENTITY error, or if the given string cannot possibly be a `shortname`, a further attempt is made using the USRFIELD attribute to construct the LDAP query.

Attention: If you have run the DEFINE AUTHINFO command, you must restart the queue manager. If you do not restart the queue manager, the setmqaut command does not return the correct result.

For processing user authorizations, the following setmqaut command settings are all equivalent.

Table 75. User authorization settings

Command	Note
`setmqaut -m QM -t qmgr -p jodoe +connect`	This is a flat, unqualified name, resolved through SHORTUSR.
`setmqaut -m QM -t qmgr -pJohnDoe1@yourcompany.com +connect`	Also a flat, unqualified name, resolving via USRFIELD to the same entity.

Table 75. User authorization settings (continued)

Command	Note
`setmqaut -m QM -t qmgr -p email=JohnDoe1@yourcompany.com +connect`	Using a named attribute.
`setmqaut -m QM -t qmgr -p "phone=1234567" +connect`	Using another named attribute which does not have to be any of those configured on the AUTHINFO object.

You can use the SET AUTHREC MQSC command as an alternative to the **setmqaut** command:

```
SET AUTHREC OBJTYPE(QMGR) PRINCIPAL('JohnDoe1@yourcompany.com') AUTHADD(connect)
```

or the Set Authority Record (MQCMD_SET_AUTH_REC) PCF command with the MQCACF_PRINCIPAL_ENTITY_NAMES element containing the string:

```
 "cn=JohnDoe,ou=users,o=yourcompany,c=yourcountry"
```

When processing groups, there is no ambiguity about `shortname` processing, as there is no requirement to fit any form of a group name into 12-characters. Therefore, there is no equivalent of the SHORTUSR attribute for groups.

That means that the syntax examples described in Table 76 on page 406 are valid, assuming that you have configured the AUTHINFO object with the extended attributes, and set to:

```
GRPFIELD(longname)
BASEDNG(ou=groups,o=yourcompany,c=yourcountry ) CLASSGRP(groupOfNames)
```

Table 76. Group authorization settings

Command	Note
`setmqaut -m QM -t qmgr -g ApplicationGroupA +connect`	Using GRPFIELD to resolve
`setmqaut -m QM -t qmgr -g longname=ApplicationGroupA +connect`	Naming a single attribute
`setmqaut -m QM -t qmgr -g "cn=Application Group A,ou=groups,o=yourcompany,c=yourcountry" +connect`	Using the full DN

You can use the SET AUTHREC MQSC command as an alternative to the preceding **setmqaut** command:

```
SET AUTHREC OBJTYPE(QMGR) GROUP('ApplicationGroupA')
      AUTHADD(connect)
```

or the Set Authority Record (MQCMD_SET_AUTH_REC) PCF command with the MQCACF_GROUP_ENTITY_NAMES element containing the string:

```
 "ApplicationGroupA"
```

Important:

Whichever format you use to refer to a name, whether for user or group, it must be possible to derive a unique DN.

So, for example, you must not have two distinct records that both have "shortu=jodoe".

If a single unique DN cannot be determined, the OAM returns MQRC_UNKNOWN_ENTITY.

Multi Displaying authorizations

Various methods of displaying authorization of users or groups.

dspmqaut command

The simplest method for displaying the authorizations available for a user or group is to use the dspmqaut command.

You can use a query on any of the syntax variations for identifying a user or group. Note that the command output repeats the identity in the format given on the command line. The output does not report on the full resolved DN.

For example:

```
dspmqaut -m QM -t qmgr -p johndoe
Entity johndoe has the following authorizations for object QM:
    connect
```

or

```
dspmqaut -m QM -t qmgr -p email=JohnDoe1@yourcompany.com
Entity email=JohnDoe1@yourcompany.com has the following authorizations for object QM:
    connect
```

dmpmqaut and dmpmqcfg commands

The dmpmqaut command, and its MQSC or PCF equivalents, can specify the principal or group in any of the supported formats, like the **setmqaut** tables described in "Setting authorizations" on page 405. However, unlike **dspmqaut**, the **dmpmqaut** command always reports the full DN.

```
dmpmqaut -m QM -t qmgr -p jodoe
------------------------------------
profile: self
object type:qmgr
entity:cn=JohnDoe, ou=users, o=yourcompany, c=yourcountry
entity type: principal
authority: connect
```

Similarly, the dmpmqcfg command, which does not have any filtering on the selected records, always shows the full DN in a format that can be replayed later.

```
dmpmqcfg -m QM -x authrec
------------------------------------
SET AUTHREC PROFILE(SELF) +
    PRINCIPAL('cn=JohnDoe, ou=users, o=yourcompany, c=yourcountry') +
    OBJTYPE(QMGR)
    AUTHADD(CONNECT)
```

Multi Other considerations when using LDAP authorization

A brief description of changes to the Message Queue Interface (MQI) and other MQSC and PCF commands that you need to be aware of when using LDAP authorization from IBM MQ 9.0.0.

ADOPTCTX

There is no requirement for applications to provide authentication information, or for the ADOPTCTX attribute to be set to YES.

If an application does not explicitly authenticate, or if **ADOPTCTX** is set to NO for the active CONNAUTH object, the identity context associated with the application is taken from the operating system user ID.

When authorizations need to be applied, that context is mapped to an LDAP identity using the same rules as for the setmqaut commands.

Input parameters to MQI calls

MQOPEN, MQPUT1, and MQSUB have structures that allow an alternative user ID to be specified.

If those fields are used, the 12-character user ID is mapped to a DN using the same rules as on the **setmqaut**, **dmpmqaut**, and **dspmqaut** commands.

MQPUT and MQPUT1 also allow suitably authorized programs to set the MQMD UserIdentifier field. The value of this field is not policed during the PUT process, and can be set to any value.

As usual, however, the **UserIdentifier** value can be used for authorization at later stages of the message processing, for example when PUTAUT(CTX) is defined on a receiving channel.

At that point, the identifier will be checked for authorization using the configuration of that receiving queue manager - which can be LDAP or OS-based.

Output parameters to MQI calls

Wherever a user ID is provided to a program in an MQI structure, it is the 12-character short name version associated with the connection.

For example, the **MQAXC.UserId** value for API Exits is the short name returned from the LDAP mapping.

Other administrative MQSC and PCF commands

Commands that show user information in object status such as DISPLAY CONN USERID return the 12-character short name associated with the context. The full DN is not shown.

Commands that allow assertion of identities, such as the CHLAUTH mapping rules or MCAUSER values for channels, can take values up to the maximum length defined for those attributes (currently 64 characters).

There is no change to the syntax. When authorization is required for that identity, it is internally mapped to a DN using the same rules as for the **setmqaut**, **dmpmqaut**, and **dspmqaut** commands.

This means that the MCAUSER value on a channel definition might not display as the same string as DISPLAY CHSTATUS but they do refer to the same identity.

For example:

```
DEFINE CHL(SV1) CHLTYPE(SVRCONN) MCAUSER('cn=JohnDoe')
DEFINE CHL(SV2) CHLTYPE(SVRCONN) MCAUSER('jodoe')
DEFINE CHL(SV3) CHLTYPE(SVRCONN) MCAUSER('JohnDoe1@yourcompany.com')
```

Then DISPLAY CHSTATUS(*) ALL shows the SHORTUSR value, *MCAUSER(jodoe)* for all connections.

Multi Switching between OS and LDAP authorization models

How you switch between the different authorization methods on different platforms.

The CONNAUTH attribute of the queue manager points at an AUTHINFO object. When the object is of type IDPWLDAP, an LDAP repository is used for authentication.

You can now apply an authorization method to that same object, which allows you to continue with OS-based authorization, or to work with LDAP authorization

IBM i, AIX and Linux

IBM i Linux AIX

The queue manager can be switched at any time between OS and LDAP models. You can change the configuration and make that configuration active by using the REFRESH SECURITY TYPE (CONNAUTH) command.

For example, if this object has already been configured with the connection information for authentication:

```
ALTER AUTHINFO(MYLDAP) AUTHTYPE(IDPWLDAP) +
        AUTHORMD(SEARCHGRP) +
        BASEDNG('ou=groups,o=ibm,c=uk') +
        <other attributes>
ALTER QMGR CONNAUTH(MYLDAP)
REFRESH SECURITY
```

Windows

Windows

If an authority configuration change involves switching between OS and LDAP models, the queue manager must be restarted for the change to take effect. Otherwise, you can make the change active by using the REFRESH SECURITY TYPE (CONNAUTH) command.

Processing rules

When switching from OS to LDAP authorization, any existing OS authority rules that have been set, become inactive and invisible.

Commands such as **dmpmqaut** do not display those OS rules. Similarly, when switching back from LDAP to OS, any defined LDAP authorizations become inactive and invisible, restoring the original OS rules.

If you want to back up the definitions of a queue manager for any reason, using the **dmpmqcfg** command, then that backup will contain only the rules that are defined for the authorization method in effect at the time of the back up.

Multi LDAP administration

An overview of how each platform administers LDAP.

When using LDAP authorization, membership of the mqm group (or equivalent) in the operating system is not that important. Being a member of that group only controls whether certain command-line commands can be processed.

In particular, you must be in that group to issue the strmqm and endmqm commands.

Once the queue manager is running, there are now limits on the fully-privileged account. Apart from the user ID of the person who issues the **strmqm** command, other users belonging to the OS mqm (or equivalent) group do not get special privileges.

Authorizations of other users are based on which LDAP groups they belong to. An unqualified use of the mqm group name in commands such as **setmqaut** is not allowed to map to any LDAP group.

AIX and Linux

Linux AIX

Once the queue manager is running, the only automatically fully-privileged account is the real user who started the queue manager.

The `mqm` ID still exists and is used as the owner of OS resources, such as files, because `mqm` is the effective ID under which the queue manager is running. However, the `mqm` user will not automatically be able to do administrative tasks controlled by the OAM.

Windows

Windows

On Windows, the automatically fully-privileged accounts are the OS user that started the queue manager, and also the user running the core queue manager processes, such as MUSR_MQADMIN if the queue manager was started as a Windows service.

When running in LDAP authorization mode, Windows behaves very similarly to the AIX and Linux platforms. It deals with 12 character short names, and full DNs.

IBM i

IBM i

On IBM i, the automatically-privileged accounts are the one that starts the queue manager and the QMQM ID.

You need both IDs, because the user ID that starts the queue manager is required only to start the system. Once running, the queue manager processes have QMQM authority only.

Sample script to provide MQADMIN privileges

Linux AIX

As it is useful to have a group able to do full administration on a queue manager, a sample script is shipped on AIX and Linux platforms as:

```
MQ_INSTALLATION_PATH/samp/bin/amqauthg.sh
```

This sample takes two parameters:

- A queue manager name
- An LDAP group name

The sample processes setmqaut commands, granting full authority for all objects. This is the same script that is generated by the IBM MQ Explorer OAM Wizard for administrative roles. For example, the code starts:

```
setmqaut -t q -m qmgr -n "**" +alldm -g
      groupname
```

Confidentiality of messages

Encrypting messages ensures that the contents of messages remains confidential. There are various methods of encrypting messages in IBM MQ depending on your needs.

If you need application-level, end-to-end data protection for your point to point messaging infrastructure, you can use Advanced Message Security to encrypt the messages, or write your own API exit or API-crossing exit.

The most secure solution is to provide end-to-end encryption, by encrypting a message from the point it is put by an application, to the point where it is got by the consuming application. This can be done using "Planning for Advanced Message Security" on page 108 (AMS) , or by writing your own API exit or API-crossing exit; see "Implementing confidentiality in user exit programs" on page 456 for more information.

If you need to encrypt messages only while they are being transported over a network, you can use TLS; see "TLS security protocols in IBM MQ " on page 24 for more information, or you can write your own security exit, message exit, or send and receive exit programs to perform encryption.

z/OS If you need to encrypt messages at rest on a queue manager, you can use z/OS data set encryption on that queue manager; see "Confidentiality for data at rest on IBM MQ for z/OS with data set encryption" on page 458 for more information.

Related tasks

Connecting two queue managers using TLS

Connecting a client to a queue manager securely

Enabling CipherSpecs

Enable a CipherSpec by using the **SSLCIPH** parameter in either the **DEFINE CHANNEL** or **ALTER CHANNEL** MQSC command.

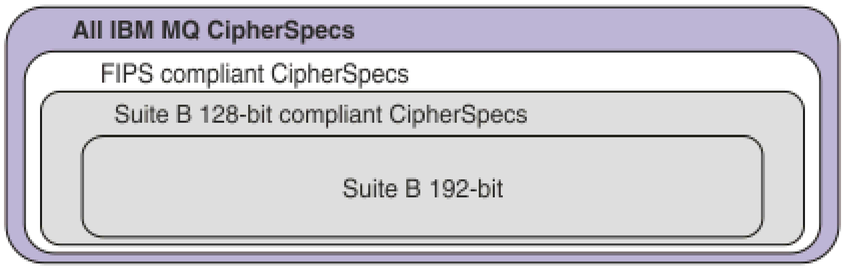
Note: On AIX, Linux, and Windows, IBM MQ provides FIPS 140-2 compliance through the IBM Crypto for C (ICC) cryptographic module. The certificate for this module has been moved to the Historical status. Customers should view the IBM Crypto for C (ICC) certificate and be aware of any advice provided by NIST. A replacement FIPS 140-3 module is currently in progress and its status can be viewed by searching for it in the NIST CMVP modules in process list.

The IBM MQ Operator 3.2.0 and queue manager container image 9.4.0.0 onwards are based on UBI 9. FIPS 140-3 compliance is currently pending and its status can be viewed by searching for "Red Hat Enterprise Linux 9 - OpenSSL FIPS Provider" in the NIST CMVP modules in process list.

Some of the CipherSpecs that you can use with IBM MQ are FIPS compliant. Some of the FIPS compliant CipherSpecs are also Suite B compliant although others, such as TLS_RSA_WITH_AES_256_CBC_SHA, are not.

All Suite B compliant CipherSpecs are also FIPS compliant. All Suite B compliant CipherSpecs fall into two groups: 128 bit (for example, ECDHE_ECDSA_AES_128_GCM_SHA256) and 192 bit (for example, ECDHE_ECDSA_AES_256_GCM_SHA384),

The following diagram illustrates the relationship between these subsets:


All IBM MQ CipherSpecs
FIPS compliant CipherSpecs
Suite B 128-bit compliant CipherSpecs
Suite B 192-bit

The product supports the TLS 1.3 security protocol on all platforms.

The CipherSpecs that you can use for each of these platforms are listed in Table 77 on page 412. For information about using these CipherSpecs, see "Using TLS 1.3 in IBM MQ" on page 414 and "IBM MQ MQI client and TLS 1.3" on page 415.

For ease of configuration and future migration, IBM MQ also provides a set of alias CipherSpecs. Migrating existing security configurations to use an alias CipherSpec means that you can adapt to cipher additions and deprecations without needing to make further invasive configuration changes in the future. These alias CipherSpecs are listed in the Alias CipherSpecs section in Table 77 on page 412. For more information about migrating to use an alias CipherSpec, see Migrating existing security configurations to use an alias CipherSpec.

You can configure the default CipherSpecs as described in "Default CipherSpec values enabled in IBM MQ" on page 415. You can also provide an alternative set of CipherSpecs that are enabled for use with channels on:

- Multi IBM MQ for Multiplatforms, as described in "Providing a custom list of ordered and enabled CipherSpecs on IBM MQ for Multiplatforms" on page 424.
- z/OS IBM MQ for z/OS, as described in "Providing a custom list of ordered and enabled CipherSpecs on IBM MQ for z/OS" on page 425.

Deprecated CipherSpecs that you can re-enable to use with IBM MQ if necessary are listed in "Deprecated CipherSpecs" on page 425.

CipherSpecs that you can use with IBM MQ TLS support

CipherSpecs that you can use with the IBM MQ queue manager automatically are listed in the following table. When you request a personal certificate, you specify a key size for the public and private key pair. The key size that is used during the TLS handshake is the size stored in the certificate unless it is determined by the CipherSpec, as noted in the table.

Table 77. CipherSpecs you can use with IBM MQ TLS support

Platform support "1" on page 414	CipherSpec name	Hex code	Protocol used	MAC algorithm	Encryption algorithm (encryption bits)	FIPS "2" on page 414	Suite B
Alias CipherSpecs							
All	ANY_TLS13_OR_HIGHER "3" on page 414 "4" on page 414	N/A	Negotiated	Negotiated	Negotiated	Negotiated	Negotiated
All	ANY_TLS13 "4" on page 414 "5" on page 414	N/A	TLS 1.3	Negotiated	Negotiated	Negotiated	Negotiated
All	ANY_TLS12_OR_HIGHER "4" on page 414 "6" on page 414	N/A	Negotiated	Negotiated	Negotiated	Negotiated	Negotiated
All	ANY_TLS12 "7" on page 414	N/A	TLS 1.2	Negotiated	Negotiated	Negotiated	Negotiated
All	ANY "8" on page 414	N/A	Negotiated	Negotiated	Negotiated	Negotiated	Negotiated
CipherSpecs for TLS 1.3							
All	TLS_AES_128_GCM_SHA256	1301	TLS 1.3	GCM	AES-128 with GCM (128)	Yes	No
All	TLS_AES_256_GCM_SHA384	1302	TLS 1.3	GCM	AES-256 with GCM (256)	Yes	No
All	TLS_CHACHA20_POLY1305_SHA256	1303	TLS 1.3	POLY1305	CHACHA20 (256)	No	No
ALW	TLS_AES_128_CCM_SHA256	1304	TLS 1.3	CBC-MAC	AES-128 with CTR (128)	Yes	No

Table 77. CipherSpecs you can use with IBM MQ TLS support (continued)

Platform support "1" on page 414	CipherSpec name	Hex code	Protocol used	MAC algorithm	Encryption algorithm (encryption bits)	FIPS "2" on page 414	Suite B
ALW	TLS_AES_128_CCM_8_SHA256 "10" on page 414	1305	TLS 1.3	CBC-MAC	AES-128 with CTR (128)	Yes	No
CipherSpecs for TLS 1.2							
All	TLS_RSA_WITH_AES_128_CBC_SHA256 "9" on page 414	003C	TLS 1.2	SHA-256	AES (128)	Yes	No
All	TLS_RSA_WITH_AES_256_CBC_SHA256 "9" on page 414 "11" on page 414	003D	TLS 1.2	SHA-256	AES (256)	Yes	No
All	TLS_RSA_WITH_AES_128_GCM_SHA256 "9" on page 414 "12" on page 414	009C	TLS 1.2	SHA-256 and AEAD GCM	AES (128)	Yes	No
All	TLS_RSA_WITH_AES_256_GCM_SHA384 "9" on page 414 "11" on page 414 "12" on page 414	009D	TLS 1.2	SHA-384 and AEAD GCM	AES (256)	Yes	No
All	ECDHE_ECDSA_AES_128_CBC_SHA256 "9" on page 414	C023	TLS 1.2	SHA-256	AES (128)	Yes	No
All	ECDHE_ECDSA_AES_256_CBC_SHA384 "9" on page 414 "11" on page 414	C024	TLS 1.2	SHA-384	AES (256)	Yes	No
All	ECDHE_RSA_AES_128_CBC_SHA256 "9" on page 414	C027	TLS 1.2	SHA-256	AES (128)	Yes	No
All	ECDHE_RSA_AES_256_CBC_SHA384 "9" on page 414 "11" on page 414	C028	TLS 1.2	SHA-384	AES (256)	Yes	No
Multi	ECDHE_ECDSA_AES_128_GCM_SHA256 "11" on page 414 "12" on page 414	C02B	TLS 1.2	SHA-256 and AEAD GCM	AES (SHA384)	Yes	128 bit
Multi	ECDHE_ECDSA_AES_256_GCM_SHA384 "11" on page 414 "12" on page 414	C02C	TLS 1.2	SHA-384 and AEAD GCM	AES (SHA384)	Yes	192 bit
All	ECDHE_RSA_AES_128_GCM_SHA256 "12" on page 414	C02F	TLS 1.2	SHA-256 and AEAD GCM	AES (128)	Yes	No
All	ECDHE_RSA_AES_256_GCM_SHA384 "11" on page 414 "12" on page 414	C030	TLS 1.2	AEAD AES-128 GCM	AES (SHA384)	Yes	No

Table 77. CipherSpecs you can use with IBM MQ TLS support (continued)

Platform support "1" on page 414	CipherSpec name	Hex code	Protocol used	MAC algorithm	Encryption algorithm (encryption bits)	FIPS "2" on page 414	Suite B

Notes:

1. For a list of platforms covered by each platform icon, see Icons used in the product documentation.
2. Specifies whether the CipherSpec is FIPS-certified on a FIPS-certified platform. See Federal Information Processing Standards (FIPS) for an explanation of FIPS.
3. ALW The ANY_TLS13_OR_HIGHER alias CipherSpec negotiates the highest level of security that the remote end will allow but will only connect using a TLS 1.3 or higher protocol.
4. IBM i To use TLS 1.3, or the ANY CipherSpec, on IBM i the underlying operating system version must support TLS 1.3. See System TLS support for TLSv1.3 for more information.
5. ALW The ANY_TLS13 alias CipherSpec represents a subset of acceptable CipherSpecs that use the TLS 1.3 protocol, as listed in this table for each platform.
6. ALW The ANY_TLS12_OR_HIGHER alias CipherSpec negotiates the highest level of security that the remote end will allow but will only connect using a TLS 1.2 or higher protocol.
7. The ANY_TLS12 CipherSpec represents a subset of acceptable CipherSpecs that use the TLS 1.2 protocol, as listed in this table for each platform.
8. ALW The ANY alias CipherSpec negotiates the highest level of security that the remote end will allow.
9. IBM i These CipherSpecs are not enabled on IBM i 7.4 systems that have System Value QSSLCSLCTL set to *OPSSYS.
10. ALW These CipherSpecs use an 8-octet Integrity Check Value (ICV) instead of a 16-octet ICV.
11. This CipherSpec cannot be used to secure a connection from the IBM MQ Explorer to a queue manager unless the appropriate unrestricted policy files are applied to the JRE used by the Explorer.
12. ALW Following a recommendation by GSKit, TLS 1.2 GCM CipherSpecs have a restriction which means that after 2ˆ24.5 TLS records are sent, using the same session key, the connection is terminated with message AMQ9288E. This GCM restriction is active, regardless of the FIPS mode being used.

    To prevent this error from happening, avoid using TLS 1.2 GCM Ciphers, enable secret key reset, or start your IBM MQ queue manager or client with the environment variable `GSK_ENFORCE_GCM_RESTRICTION=GSK_FALSE` set. For GSKit libraries, you must set this environment variable on both sides of the connection, and apply it to both client to queue manager connections and queue manager to queue manager connections. Note that this setting affects unmanaged .NET clients, but not Java or managed .NET clients. For more information, see AES-GCM cipher restriction.

    z/OS This restriction does not apply to IBM MQ for z/OS.

Using TLS 1.3 in IBM MQ

The product supports TLS 1.3 on all platforms.

Queue managers that are created at IBM MQ 9.2.0 or later support TLS 1.3 by default. Queue managers migrated from earlier versions of IBM MQ need to have TLS 1.3 enabled. You can enable TLS 1.3 on migrated queue managers by setting the **AllowTLSV13**=TRUE property:

- Multi For IBM MQ for Multiplatforms queue managers, edit the `qm.ini` file and add the **`AllowTLSV13`**=TRUE property under the SSL stanza (link to

  ```
  SSL:
    AllowTLSV13=TRUE
  ```

- z/OS For IBM MQ for z/OS queue managers, edit the QMINI data set specified in the queue manager startup JCL and add the **`AllowTLSV13`**=TRUE property under the TransportSecurity stanza

  ```
  TransportSecurity:
      AllowTLSV13=TRUE
  ```

When TLS 1.3 is enabled, and in accordance with the TLS 1.3 specification, any attempt to communicate with a weak CipherSpec, regardless of whether they are enabled in IBM MQ or not, is rejected. The CipherSpecs that TLS 1.3 considers weak are CipherSpecs that meet one or more of the following criteria:

- Uses the SSL 3.0 protocol.
- Uses RC4 or RC2 as the Encryption algorithm.
- Has a encryption key size (bit) equal to or less than 112.

These restrictions are flagged with Note [3] in Table 1 of Deprecated CipherSpecs.

If you need to continue using such CipherSpecs, then you must disable TLS 1.3 mode:

- ALW Edit the queue manager's `qm.ini` file and change the setting of the **`AllowTLSV13`** property to:

  ```
  SSL:
    AllowTLSV13=FALSE
  ```

- z/OS Edit the QMINI data set of the queue manager and change the setting of the **`AllowTLSV13`** property to:

  ```
  TransportSecurity:
      AllowTLSV13=FALSE
  ```

IBM MQ MQI client and TLS 1.3

ALW

When using the IBM MQ MQI client, the value of **`AllowTLSV13`** is inferred unless it is explicitly specified in the SSL stanza of the `mqclient.ini` file that is being used by the application.

- If any weak CipherSpecs are enabled, **`AllowTLSV13`** is set to FALSE and no TLS 1.3 CipherSpecs can be used.
- Otherwise, **`AllowTLSV13`** is set to TRUE and the new TLS 1.3 CipherSpecs and alias CipherSpecs can be used.

Default CipherSpec values enabled in IBM MQ

In default configuration for a new IBM MQ queue manager, IBM MQ provides support for the TLS 1.2 and TLS 1.3 protocols and various cryptographic algorithms using CipherSpecs. For compatibility purposes, IBM MQ can also be configured to use SSL 3.0 and TLS 1.0 protocols and a number of cryptographic algorithms that are known to be weak or susceptible to security vulnerabilities. The list of CipherSpecs that are enabled in default configuration might change by applying maintenance.

It is possible to configure IBM MQ to restrict or permit the use of CipherSpecs using the following controls:

- Only permit FIPS 140-2 compliant CipherSpecs using SSLFIPS.

- ALW Only permit NSA Suite B compliant CipherSpecs using SUITEB.
- Multi Permit a custom list of CipherSpecs using **`AllowedCipherSpecs`**.
- ALW Permit a custom list of CipherSpecs using the **`AMQ_ALLOWED_CIPHERS`** environment variable.
- ALW Permit the use of deprecated CipherSpecs using **`AllowWeakCipher`** or the **`AMQ_SSL_WEAK_CIPHER_ENABLE`** environment variable.
- z/OS Permit the use of deprecated CipherSpecs using DD statements in the CHINIT JCL.

Note: If you specify a custom list of CipherSpecs using **`AllowedCipherSpecs`** or **`AMQ_ALLOWED_CIPHERS`** this overrides enablement of any deprecated CipherSpecs. Note that when using either NSA Suite B or FIPS 140-2 restrictions in combination with a custom CipherSpec list, you must ensure the custom list only contains CipherSpecs permitted by the Suite B or FIPS 140-2 settings.

Related concepts

"Digital certificates and CipherSpec compatibility in IBM MQ" on page 47
This topic provides information on how to choose appropriate CipherSpecs and digital certificates for your security policy, by outlining the relationship between CipherSpecs and digital certificates in IBM MQ.

"CipherSpecs and CipherSuites" on page 22
Cryptographic security protocols must agree on the algorithms used by a secure connection. CipherSpecs and CipherSuites define specific combinations of algorithms.

"Configuring IBM MQ for Suite B" on page 44
IBM MQ can be configured to operate in compliance with the NSA Suite B standard on AIX, Linux, and Windows platforms.

"Federal Information Processing Standards (FIPS)" on page 34
This topic introduces the Federal Information Processing Standards (FIPS) Cryptomodule Validation Program of the US National Institute of Standards and Technology and the cryptographic functions which can be used on TLS channels.

Related tasks

Migrating existing security configurations to use an alias CipherSpe

Related reference

DEFINE CHANNEL

ALTER CHANNEL

Change, Copy, and Create Channel

ALW AES-GCM cipher restriction

A guide to the restrictions that are imposed on AES-GCM ciphers when used for TLS Cryptography. These restrictions are imposed by the IETF and NIST organizations and require that the same session key must not be used to securely transfer more than $2^{24.5}$ TLS records when using AES-GCM ciphers.

For more information about these restrictions, see RFC 9325 Section 4.4 Limits on Key Usage and RFC 8446 section 5.5.

IBM MQ does not implement cryptographic functionality directly. Instead, several different cryptographic libraries are used to provide TLS and Advanced Message Security functionality. On Windows, Linux, and AIX operating systems, the cryptographic library that IBM MQ uses is IBM Global Security Kit (GSKit). For applications, the C and unmanaged .NET libraries use GSKit for cryptographic functionality. The implementation of the AES-GCM encryption algorithms by GSKit includes the restrictions that are specified by the standards group. Also, these restrictions are enabled by default. As such, IBM MQ TLS communication, when using AES-GCM ciphers, terminates if more than $2^{24.5}$ TLS records are transmitted using the same session key.

Note: This restriction is not present on IBM i, IBM Z or IBM MQ for HPE NonStop platforms or Java/JMS, managed .NET applications because different cryptographic libraries are used, and these libraries have not implemented the same restriction.

If an IBM MQ channel remains running for long enough that more than $2^{24.5}$ TLS records are transmitted using the same session key, the underlying cryptographic library terminates the connection. This causes the channel to terminate and an AMQ9288E error message is generated. Applications that have their communication terminated in this way receive an `MQRC_CONNECTION_BROKEN` return code from whichever IBM MQ operation was being performed.

The termination of the connection can be performed at either end of the communication, but only on ends that are using GSKit for cryptographic functionality.

Advice for mitigating the restriction

Some options for how to prevent or handle communications that are terminated due to this restriction are as follows:

Use reconnectable clients

Applications can be configured to automatically attempt a reconnection, should a connection fail. This includes connections that are terminated due to the GCM restriction. When configured for reconnection, the client application is restored automatically at any point of failure and any handles to open objects are restored. This is done without returning to the application code.

For more information, see Automatic client reconnection.

Set a secret key reset value

IBM MQ can be configured to request a session key reset after a configurable number of bytes has been transferred over a channel. Upon reaching this limit, IBM MQ requests that the cryptographic layer performs a session key reset, resulting in a new session key.

It is important to note that the value specified is the number of bytes transferred, which relates to the size of the messages being sent by IBM MQ. The restriction is on the number of TLS records that are sent. There is not a direct mapping between message bytes and TLS records as a TLS record can send a maximum number of bytes dependent on the Maximum Transmission Unit (MTU) of the network. Any messages that are sent that are larger than this value are transmitted as multiple TLS records. The MTU value varies between networks. Also, there are other reasons why a TLS record might need to be sent outside of transmitting IBM MQ message data, for example IBM MQ Heartbeat checks, TLS alerts, other IBM MQ protocol messages. These additional TLS records count toward the maximum number of TLS records but are not counted in the IBM MQ secret key reset value.

Regularly resetting a session key using secret key reset can prevent the channel from being terminated due to the AES-GCM restriction.

For more information, see Resetting SSL and TLS secret keys.

Use TLS 1.3 cipherspecs

While the AES-GCM restriction is still present when using the TLS 1.3 protocol, the TLS 1.3 protocol supports automatically performing a session key reset without the need to interrupt TLS communications. This allows GSKit to manage resetting the session key when it is necessary without IBM MQ needing to request a secret key reset.

For more information, see Using TLS 1.3 in IBM MQ in “Enabling CipherSpecs” on page 411.

Disable the AES-GCM restriction

If needed, the restriction can be disabled by setting the environment variable **`GSK_ENFORCE_GCM_RESTRICTION=GSK_FALSE`** to disable the AES-GCM restriction. Doing so allows any number of TLS records to be sent using the same session key. If choosing this mitigation, the environment variable must be set on each end of communication that uses GSKit for secure communications.

Warning: This option is not recommended as, after more than $2^{24.5}$ TLS records have been sent, it is possible for attackers to perform analysis on the sent records to determine the session key in use. Once the session key has been determined, all existing and future communication using that session key is compromised.

CipherSpec order in TLS handshake

The order of CipherSpecs is used when choosing between multiple possible CipherSpecs, for example when using one of the ANY* CipherSpecs.

During a TLS handshake, a client and server exchange the CipherSpecs and protocols that they support in order of their preference. A common CipherSpec that both sides prioritize is chosen and used for the TLS communication. When choosing a CipherSpec protocol, version is also considered, for example if a server lists TLS 1.2 CipherSpecs before TLS 1.3 CipherSpecs it will still prioritize TLS 1.3 so long as the client can support it and has a common TLS 1.3 CipherSpec that can be used.

When IBM MQ is configured for TLS it sets the CipherSpecs into the order shown in the following table, from most preferred to least preferred.

Note: If a CipherSpec is not enabled through the **`AllowedCipherSpecs`** attribute, it will not be configured for use during a TLS Handshake.

In the case that the **`AllowedCipherSpecs`** attribute is not specified, a default list of enabled ciphers, indicated by the following table, is used.

Table 78. CipherSpecs order from IBM MQ 9.2.0

Platform	CipherSpec	Protocol	Hexadecimal code	Enabled by default
All	TLS_CHACHA20_POLY1305_SHA256	TLS 1.3	1303	Yes
All	TLS_AES_256_GCM_SHA384	TLS 1.3	1302	Yes
All	TLS_AES_128_GCM_SHA256	TLS 1.3	1301	Yes
ALW	TLS_AES_128_CCM_SHA256	TLS 1.3	1304	Yes
ALW	TLS_AES_128_CCM_8_SHA256	TLS 1.3	1305	Yes
All	TLS_RSA_WITH_AES_256_GCM_SHA384	TLS 1.2	009D	Yes
Multi	ECDHE_ECDSA_AES_256_GCM_SHA384	TLS 1.2	C02C	Yes
All	ECDHE_RSA_AES_256_GCM_SHA384	TLS 1.2	C030	Yes
All	TLS_RSA_WITH_AES_256_CBC_SHA256	TLS 1.2	003D	Yes
All	ECDHE_ECDSA_AES_256_CBC_SHA384	TLS 1.2	C024	Yes
All	ECDHE_RSA_AES_256_CBC_SHA384	TLS 1.2	C028	Yes

Table 78. CipherSpecs order from IBM MQ 9.2.0 (continued)

Platform	CipherSpec	Protocol	Hexadecimal code	Enabled by default
All	TLS_RSA_WITH_AES_128_GCM_SHA256	TLS 1.2	009C	Yes
Multi	ECDHE_ECDSA_AES_128_GCM_SHA256	TLS 1.2	C02B	Yes
All	ECDHE_RSA_AES_128_GCM_SHA256	TLS 1.2	C02F	Yes
All	TLS_RSA_WITH_AES_128_CBC_SHA256	TLS 1.2	003C	Yes
All	ECDHE_ECDSA_AES_128_CBC_SHA256	TLS 1.2	C023	Yes
All	ECDHE_RSA_AES_128_CBC_SHA256	TLS 1.2	C027	Yes
ALW	ECDHE_ECDSA_3DES_EDE_CBC_SHA256	TLS 1.2	C008	No
Multi	ECDHE_RSA_3DES_EDE_CBC_SHA256	TLS 1.2	C012	No
ALW	TLS_RSA_WITH_RC4_128_SHA256	TLS 1.2	0005	No
ALW	ECDHE_ECDSA_RC4_128_SHA256	TLS 1.2	C007	No
Multi	ECDHE_RSA_RC4_128_SHA256	TLS 1.2	C011	No
All	TLS_RSA_WITH_NULL_SHA256	TLS 1.2	003B	No
ALW	ECDHE_ECDSA_NULL_SHA256	TLS 1.2	C006	No
Multi	ECDHE_RSA_NULL_SHA256	TLS 1.2	C010	No
ALW	TLS_RSA_WITH_NULL_NULL	TLS 1.2	0000	No
ALW z/OS	TLS_RSA_WITH_AES_256_CBC_SHA	TLS 1.0	0035	No
ALW z/OS	TLS_RSA_WITH_AES_128_CBC_SHA	TLS 1.0	002F	No

Table 78. CipherSpecs order from IBM MQ 9.2.0 (continued)

Platform	CipherSpec	Protocol	Hexadecimal code	Enabled by default
IBM i	AES_SHA_US	TLS 1.0	002E	No
All	TLS_RSA_WITH_3DES_EDE_CBC_SHA	TLS 1.0	000A	No
All	TLS_RSA_WITH_RC4_128_SHA	TLS 1.0	0005	No
IBM i	TLS_RSA_WITH_RC4_128_MD5	TLS 1.0	0004	No
All	TLS_RSA_WITH_DES_CBC_SHA	TLS 1.0	0009	No
IBM i	TLS_RSA_EXPORT_WITH_RC4_40_MD5	TLS 1.0	0003	No
IBM i	TLS_RSA_EXPORT_WITH_RC2_40_MD5	TLS 1.0	0006	No
IBM i	TLS_RSA_WITH_NULL_SHA	TLS 1.0	0002	No
IBM i	TLS_RSA_WITH_NULL_MD5	TLS 1.0	0001	No
All	TRIPLE_DES_SHA_US	SSL v3	000A	No
All	RC4_SHA_US	SSL v3	0005	No
All	RC4_MD5_US	SSL v3	0004	No
All	DES_SHA_EXPORT	SSL v3	0009	No
All	RC4_MD5_EXPORT	SSL v3	0003	No
All	RC2_MD5_EXPORT	SSL v3	0006	No
All	NULL_SHA	SSL v3	0002	No
All	NULL_MD5	SSL v3	0001	No
ALW	FIPS_WITH_3DES_EDE_CBC_SHA	SSL v3	FEFF	No
ALW	RC4_56_SHA_EXPORT1024	SSL v3	0064	No
ALW	DES_SHA_EXPORT1024	SSL v3	0062	No
ALW	FIPS_WITH_DES_CBC_SHA	SSL v3	FEFE	No

This list was constructed by ordering the protocols with the default list supplied by the cryptographic library used by IBM MQ on z/OS and is consistent across z/OS and distributed platforms.

Changing the order

If a different order is desired, then a new order of CipherSpecs can be supplied using the **AllowedCipherSpecs** attribute of the SSL stanza on IBM MQ for Multiplatforms z/OS, or the TransportSecurity stanza on IBM MQ for z/OS, with the following rules:

- Higher protocol versions are always used, regardless of their position in the list.
- Any disabled CipherSpecs are re-enabled if supplied in the list.
- The TLS server's list order has a higher priority than the TLS client.
- When TLS 1.3 is enabled, certain CipherSpecs are not supported.

For example, on IBM MQ for Multiplatforms, if the following is configured on the queue manager:

```
SSL:
AllowedCipherSpecs=TLS_RSA_WITH_AES_128_GCM_SHA256,TLS_AES_128_GCM_SHA256,
TLS_AES_256_GCM_SHA384,TLS_RSA_WITH_AES_256_GCM_SHA384,TLS_RSA_WITH_AES_256_CBC_SHA
```

z/OS and on IBM MQ for z/OS, if the following is configured on the queue manager:

```
TransportSecurity:
AllowedCipherSpecs=TLS_RSA_WITH_AES_128_GCM_SHA256,TLS_AES_128_GCM_SHA256,
TLS_AES_256_GCM_SHA384,TLS_RSA_WITH_AES_256_GCM_SHA384,TLS_RSA_WITH_AES_256_CBC_SHA
```

then:

- A client connecting with ANY_TLS12 will likely use the TLS 1.2 CipherSpec TLS_RSA_WITH_AES_128_GCM_SHA256.
- A client connecting with ANY_TLS12_OR_HIGHER will likely use the TLS 1.3 CipherSpec TLS_AES_128_GCM_SHA256 (assuming the client supports TLS 1.3).
- A client connecting with the TLS 1.0 CipherSpec TLS_RSA_WITH_AES_256_CBC_SHA will use that CipherSpec.

Previous versions of IBM MQ

Before IBM MQ 9.2.0, the following order of CipherSpecs was used:

Table 79. CipherSpecs order before IBM MQ 9.2.0

Platform	CipherSpec	Protocol	Enabled by default
ALW z/OS	TLS_RSA_WITH_AES_128_CBC_SHA	TLS 1.0	No
IBM i	AES_SHA_US	TLS 1.0	No
ALW z/OS	TLS_RSA_WITH_AES_256_CBC_SHA	TLS 1.0	No
All	RC4_SHA_US	SSL v3	No
All	TLS_RSA_WITH_RC4_128_SHA	TLS 1.0	No
All	RC4_MD5_US	SSL v3	No
IBM i	TLS_RSA_WITH_RC4_128_MD5	TLS 1.0	No

Table 79. CipherSpecs order before IBM MQ 9.2.0 (continued)

Platform	CipherSpec	Protocol	Enabled by default
All	TRIPLE_DES_SHA_US	SSL v3	No
All	TLS_RSA_WITH_3DES_EDE_CBC_SHA	TLS 1.0	No
ALW	DES_SHA_EXPORT1024	SSL v3	No
All	RC4_56_SHA_EXPORT1024	SSL v3	No
All	RC4_MD5_EXPORT	SSL v3	No
IBM i	TLS_RSA_EXPORT_WITH_RC4_40_MD5	TLS 1.0	No
All	RC2_MD5_EXPORT	SSL v3	No
IBM i	TLS_RSA_EXPORT_WITH_RC2_40_MD5	TLS 1.0	No
All	DES_SHA_EXPORT	SSL v3	No
All	TLS_RSA_WITH_DES_CBC_SHA	TLS 1.0	No
All	NULL_SHA	SSL v3	No
IBM i	TLS_RSA_WITH_NULL_SHA	TLS 1.0	No
All	NULL_MD5	SSL v3	No
IBM i	TLS_RSA_WITH_NULL_MD5	TLS 1.0	No
ALW	FIPS_WITH_DES_CBC_SHA	SSL v3	No
ALW	FIPS_WITH_3DES_EDE_CBC_SHA	SSL v3	No
All	TLS_RSA_WITH_AES_128_CBC_SHA256	TLS 1.2	Yes
All	TLS_RSA_WITH_AES_256_CBC_SHA256	TLS 1.2	Yes
All	TLS_RSA_WITH_NULL_SHA256	TLS 1.2	No
All	TLS_RSA_WITH_AES_128_GCM_SHA256	TLS 1.2	Yes
All	TLS_RSA_WITH_AES_256_GCM_SHA384	TLS 1.2	Yes
ALW	ECDHE_ECDSA_RC4_128_SHA256	TLS 1.2	No
ALW	ECDHE_ECDSA_3DES_EDE_CBC_SHA256	TLS 1.2	No

Platform	CipherSpec	Protocol	Enabled by default
Multi	ECDHE_RSA_RC4_128_SHA256	TLS 1.2	No
Multi	ECDHE_RSA_3DES_EDE_CBC_SHA256	TLS 1.2	No
All	ECDHE_ECDSA_AES_128_CBC_SHA256	TLS 1.2	Yes
All	ECDHE_ECDSA_AES_256_CBC_SHA384	TLS 1.2	Yes
All	ECDHE_RSA_AES_128_CBC_SHA256	TLS 1.2	Yes
All	ECDHE_RSA_AES_256_CBC_SHA384	TLS 1.2	Yes
Multi	ECDHE_ECDSA_AES_128_GCM_SHA256	TLS 1.2	Yes
Multi	ECDHE_ECDSA_AES_256_GCM_SHA384	TLS 1.2	Yes
All	ECDHE_RSA_AES_128_GCM_SHA256	TLS 1.2	Yes
All	ECDHE_RSA_AES_256_GCM_SHA384	TLS 1.2	Yes
Multi	ECDHE_RSA_NULL_SHA256	TLS 1.2	No
ALW	ECDHE_ECDSA_NULL_SHA256	TLS 1.2	No
ALW	TLS_RSA_WITH_NULL_NULL	TLS 1.2	No
ALW	TLS_RSA_WITH_RC4_128_SHA256	TLS 1.2	No
Multi	TLS_AES_128_GCM_SHA256	TLS 1.3	Yes
Multi	TLS_AES_256_GCM_SHA384	TLS 1.3	Yes
Multi	TLS_CHACHA20_POLY1305_SHA256	TLS 1.3	Yes
ALW	TLS_AES_128_CCM_SHA256	TLS 1.3	Yes
ALW	TLS_AES_128_CCM_8_SHA256	TLS 1.3	Yes

Table 79. CipherSpecs order before IBM MQ 9.2.0 (continued)

Important: As of 23rd July 2020, the following AllowedCipherSpecs attribute only enables CipherSpecs that are currently enabled by default. However, you should verify the CipherSpecs enabled by the following AllowedCipherSpecs attribute with current data, to ensure that CipherSpecs that have been deprecated since this date are not inadvertently re-enabled.

If you need to return to this order of CipherSpecs, you can do so by using the following **AllowedCipherSpecs** SSL/TransportSecurity stanza attribute value:

```
AllowedCipherSpecs=TLS_RSA_WITH_AES_128_CBC_SHA256,TLS_RSA_WITH_AES_256_CBC_SHA256,
TLS_RSA_WITH_AES_128_GCM_SHA256,TLS_RSA_WITH_AES_256_GCM_SHA384,ECDHE_ECDSA_AES_128_CBC_SHA256,
ECDHE_ECDSA_AES_256_CBC_SHA384,ECDHE_RSA_AES_128_CBC_SHA256,ECDHE_RSA_AES_256_CBC_SHA384,
ECDHE_ECDSA_AES_128_GCM_SHA256,ECDHE_ECDSA_AES_256_GCM_SHA384,ECDHE_RSA_AES_128_GCM_SHA256,
ECDHE_RSA_AES_256_GCM_SHA384
```

Providing a custom list of ordered and enabled CipherSpecs on IBM MQ for Multiplatforms

Multi

You can provide an alternative set of CipherSpecs that are enabled, and in your order of preference, for use with IBM MQ channels, either using the ALW **AMQ_ALLOWED_CIPHERS** environment variable or the **AllowedCipherSpecs** SSL stanza attribute of the `.ini` file. You might want to use this setting for either of the following reasons:

- To restrict IBM MQ listeners from accepting incoming channel start requests, unless they use one of the named CipherSpecs.
- To change the order of priority of CipherSpecs that are used in a TLS handshake.

This functionality can be used to control the CipherSpecs that are included in the ANY* CipherSpecs.

The **AMQ_ALLOWED_CIPHERS** environment variable or **AllowedCipherSpecs** SSL stanza attribute accepts:

- A single CipherSpec name.
- A comma separated list of CipherSpec names to re-enable.
- The special value of ALL, representing all CipherSpecs.

Note: You should not enable **ALL** CipherSpecs, as this will enable SSL 3.0 and TLS 1.0 protocols and a large number of weak cryptographic algorithms.

If this setting is configured, it overrides the default CipherSpec list and causes IBM MQ to ignore weak cipher deprecation settings (see below):

- IBM MQ listeners only accept SSL/TLS proposals that use one of the named CipherSpecs.
- IBM MQ channels only allow a blank SSLCIPH value, or one of the named CipherSpecs.
- **runmqsc** tab completion of SSLCIPH values restricts the completion values to one of the name CipherSpecs.

For example, if you only want to allow channels to be defined/altered and listeners to accept ECDHE_RSA_AES_128_GCM_SHA256 or ECDHE_ECDSA_AES_256_GCM_SHA384 you could set the following in the `qm.ini` file:

```

SSL:
  AllowedCipherSpecs=ECDHE_RSA_AES_128_GCM_SHA256, ECDHE_ECDSA_AES_256_GCM_SHA384
```

Additionally, the CipherSpecs in this list will be used to determine the priority of the CipherSpecs used during a TLS handshake. For example, if you specify a list of `TLS_RSA_WITH_AES_128_CBC_SHA256, TLS_RSA_WITH_AES_256_CBC_SHA256` it is likely that, during the handshake, the TLS_RSA_WITH_AES_128_CBC_SHA256 CipherSpec will be chosen over the TLS_RSA_WITH_AES_256_CBC_SHA256 CipherSpec if a client connects specifying both of these CipherSpecs, that is, a client connecting with ANY_TLS12.

Note that ciphers used by AMQP or MQTT channels can be restricted using java.security file settings.

Providing a custom list of ordered and enabled CipherSpecs on IBM MQ for z/OS

z/OS

It is possible for you to provide an alternative set of CipherSpecs that are enabled, and in your order of preference, for use with IBM MQ channels, using the **AllowedCipherSpecs** TransportSecurity stanza attribute of The QMINI data set. You might want to do this for either of the following reasons:

- To restrict IBM MQ listeners from accepting incoming channel start requests, unless they use one of the named CipherSpecs.
- To change the order of priority of CipherSpecs that are used in a TLS handshake.

You can use this functionality to control the CipherSpecs that are included in the ANY* CipherSpecs. The **AllowedCipherSpecs** attribute accepts:

- A single CipherSpec name.
- A comma separated list of CipherSpec names to re-enable.
- The special value of ALL, representing all CipherSpecs.

  Note: You should not enable **ALL** CipherSpecs, as this will enable SSL 3.0 and TLS 1.0 protocols and a large number of weak cryptographic algorithms. If you do configure this setting, it overrides the default CipherSpec list and causes IBM MQ to ignore weak cipher deprecation settings; see "Enabling deprecated CipherSpecs on z/OS" on page 429.

IBM MQ listeners only accept SSL/TLS proposals that use one of the named CipherSpecs and IBM MQ channels only allow a blank SSLCIPH value, or one of the named CipherSpecs.

For example, if you only want to allow channels to be defined/altered and listeners to accept ECDHE_RSA_AES_128_GCM_SHA256 or ECDHE_RSA_AES_256_GCM_SHA384 you could set the following:

```
TransportSecurity:
  AllowedCipherSpecs=ECDHE_RSA_AES_128_GCM_SHA256,
                     ECDHE_RSA_AES_256_GCM_SHA384
```

Additionally, the CipherSpecs in this list are used to determine the priority of the CipherSpecs used during a TLS handshake. For example, if you specify a list of TLS_RSA_WITH_AES_128_CBC_SHA256, TLS_RSA_WITH_AES_256_CBC_SHA256 it is likely that, during the handshake, the TLS_RSA_WITH_AES_128_CBC_SHA256 CipherSpec will be chosen over the TLS_RSA_WITH_AES_256_CBC_SHA256 CipherSpec if a client connects specifying both of these CipherSpecs, that is, a client connecting with ANY_TLS12.

Deprecated Deprecated CipherSpecs

A list of deprecated CipherSpecs that you are able to use with IBM MQ if necessary.

Deprecated CipherSpecs that you can use with IBM MQ TLS support are listed in the following table.

Table 80. Deprecated CipherSpecs you can re-enable for use with IBM MQ

Platform support "1" on page 428	CipherSpec name	Hex code	Protocol used	Data integrity	Encryption algorithm (encryption bits)	FIPS "2" on page 428	Suite B	Update when deprecated
CipherSpecs for SSL 3.0								
IBM i	AES_SHA_US "3" on page 428	002F	SSL 3.0	SHA-1	AES (128)	No	No	9.0.0.0

Table 80. Deprecated CipherSpecs you can re-enable for use with IBM MQ (continued)

Platform support “1” on page 428	CipherSpec name	Hex code	Protocol used	Data integrity	Encryption algorithm (encryption bits)	FIPS “2” on page 428	Suite B	Update when deprecated
All	DES_SHA_EXPORT “3” on page 428 “4” on page 428 “5” on page 428	0009	SSL 3.0	SHA-1	DES (56)	No	No	9.0.0.0
ALW	DES_SHA_EXPORT1024 “3” on page 428 “6” on page 428	0062	SSL 3.0	SHA-1	DES (56)	No	No	9.0.0.0
ALW	FIPS_WITH_DES_CBC_SHA “3” on page 428	FEFE	SSL 3.0	SHA-1	DES (56)	No“7” on page 428	No	9.0.0.0
ALW	FIPS_WITH_3DES_EDE_CBC_SHA “3” on page 428	FEFF	SSL 3.0	SHA-1	3DES (168)	No“8” on page 428	No	9.0.0.1 and 9.0.1
All	NULL_MD5 “3” on page 428	0001	SSL 3.0	MD5	None	No	No	9.0.0.1
All	NULL_SHA “3” on page 428	0002	SSL 3.0	SHA-1	None	No	No	9.0.0.1
All	RC2_MD5_EXPORT “3” on page 428 “4” on page 428 “5” on page 428	0006	SSL 3.0	MD5	RC2 (40)	No	No	9.0.0.0
All	RC4_MD5_EXPORT “4” on page 428 “3” on page 428	0003	SSL 3.0	MD5	RC4 (40)	No	No	9.0.0.0
All	RC4_MD5_US “3” on page 428	0004	SSL 3.0	MD5	RC4 (128)	No	No	9.0.0.0
All	RC4_SHA_US “3” on page 428 “5” on page 428	0005	SSL 3.0	SHA-1	RC4 (128)	No	No	9.0.0.0
ALW	RC4_56_SHA_EXPORT1024 “3” on page 428 “6” on page 428	0064	SSL 3.0	SHA-1	RC4 (56)	No	No	9.0.0.0
All	TRIPLE_DES_SHA_US “3” on page 428 “5” on page 428	000A	SSL 3.0	SHA-1	3DES (168)	No	No	9.0.0.1 and 9.0.1
CipherSpecs for TLS 1.0								
IBM i	TLS_RSA_EXPORT_WITH_RC2_40_MD5 “3” on page 428	0006	TLS 1.0	MD5	RC2 (40)	No	No	9.0.0.0
IBM i	TLS_RSA_EXPORT_WITH_RC4_40_MD5“3” on page 428 “4” on page 428	0003	TLS 1.0	MD5	RC4 (40)	No	No	9.0.0.0
All	TLS_RSA_WITH_DES_CBC_SHA “3” on page 428	0009	TLS 1.0	SHA-1	DES (56)	No“9” on page 428	No	9.0.0.0
IBM i	TLS_RSA_WITH_NULL_MD5 “3” on page 428	0001	TLS 1.0	MD5	None	No	No	9.0.0.1

Table 80. Deprecated CipherSpecs you can re-enable for use with IBM MQ (continued)

Platform support "1" on page 428	CipherSpec name	Hex code	Protocol used	Data integrity	Encryption algorithm (encryption bits)	FIPS "2" on page 428	Suite B	Update when deprecated
IBM i	TLS_RSA_WITH_NULL_SHA "3" on page 428	0002	TLS 1.0	SHA-1	None	No	No	9.0.0.1
IBM i	TLS_RSA_WITH_RC4_128_MD5 "3" on page 428	0004	TLS 1.0	MD5	RC4 (128)	No	No	9.0.0.0
z/OS ALW	TLS_RSA_WITH_AES_128_CBC_SHA "10" on page 428	002F	TLS 1.0	SHA-1	AES (128)	Yes	No	9.0.5
z/OS ALW	TLS_RSA_WITH_AES_256_CBC_SHA "6" on page 428 "10" on page 428	0035	TLS 1.0	SHA-1	AES (256)	Yes	No	9.0.5
All	TLS_RSA_WITH_3DES_EDE_CBC_SHA	000A	TLS 1.0	SHA-1	3DES (168)	Yes	No	9.0.0.1 and 9.0.1
CipherSpecs for TLS 1.2								
ALW	ECDHE_ECDSA_NULL_SHA256 "3" on page 428	C006	TLS 1.2	SHA-1	None	No	No	9.0.0.1
ALW	ECDHE_ECDSA_RC4_128_SHA256 "3" on page 428	C007	TLS 1.2	SHA-1	RC4 (128)	No	No	9.0.0.0
ALW IBM i	ECDHE_RSA_NULL_SHA256 "3" on page 428	C010	TLS 1.2	SHA-1	None	No	No	9.0.0.1
ALW IBM i	ECDHE_RSA_RC4_128_SHA256 "3" on page 428	C011	TLS 1.2	SHA-1	RC4 (128)	No	No	9.0.0.0
ALW	TLS_RSA_WITH_NULL_NULL "3" on page 428	0000	TLS 1.2	None	None	No	No	9.0.0.1
All	TLS_RSA_WITH_NULL_SHA256 "3" on page 428	003B	TLS 1.2	SHA-256	None	No	No	9.0.0.1
ALW	TLS_RSA_WITH_RC4_128_SHA256 "3" on page 428	0005	TLS 1.2	SHA-1	RC4 (128)	No	No	9.0.0.0
ALW	ECDHE_ECDSA_3DES_EDE_CBC_SHA256	C0008	TLS 1.2	SHA-1	3DES (168)	Yes	No	9.0.0.1 and 9.0.1
ALW IBM i	ECDHE_RSA_3DES_EDE_CBC_SHA256	C012	TLS 1.2	SHA-1	3DES (168)	Yes	No	9.0.0.1 and 9.0.1

Table 80. Deprecated CipherSpecs you can re-enable for use with IBM MQ (continued)

Platform support "1" on page 428	CipherSpec name	Hex code	Protocol used	Data integrity	Encryption algorithm (encryption bits)	FIPS "2" on page 428	Suite B	Update when deprecated

Notes:

1. For a list of platforms covered by each platform icon, see Icons used in the product documentation.
2. Specifies whether the CipherSpec is FIPS-certified on a FIPS-certified platform. See Federal Information Processing Standards (FIPS) for an explanation of FIPS.
3. ALW These CipherSpecs are disabled when TLS 1.3 is enabled (through the AllowTLSV13 property in the `qm.ini`).

   z/OS Queue managers created at IBM MQ for z/OS 9.2.0 or later enable TLS 1.3 by default, which disables these CipherSpecs. You can enable these CipherSpecs, if required, by turning off TLS V1.3. This is done by adding **`AllowTLSV13`**=*FALSE* to the TransportSecurity stanza of the QMINI data set in the queue manager JCL. Queue managers migrated to IBM MQ for z/OS 9.2.0 from an earlier version don't have TLS 1.3 enabled by default, and therefore have these CipherSpecs enabled.
4. The maximum handshake key size is 512 bits. If either of the certificates exchanged during the SSL handshake has a key size greater than 512 bits, a temporary 512-bit key is generated for use during the handshake.
5. These CipherSpecs are no longer supported by IBM MQ classes for Java or IBM MQ classes for JMS. For more information, see SSL/TLS CipherSpecs and CipherSuites in IBM MQ classes for Java or SSL/TLS CipherSpecs and CipherSuites in IBM MQ classes for JMS.
6. The handshake key size is 1024 bits.
7. Deprecated This CipherSpec was FIPS 140-2 certified before 19 May 2007. The name `FIPS_WITH_DES_CBC_SHA` is historical and reflects the fact that this CipherSpec was previously (but is no longer) FIPS-compliant. This CipherSpec is deprecated and its use is not recommended.
8. Deprecated The name `FIPS_WITH_3DES_EDE_CBC_SHA` is historical and reflects the fact that this CipherSpec was previously (but is no longer) FIPS-compliant. The use of this CipherSpec is deprecated.
9. This CipherSpec was FIPS 140-2 certified before 19 May 2007.
10. Re-enabling just these CipherSpecs does not require the use of the CSQXWEAK DD statement.

Enabling deprecated CipherSpecs on IBM MQ for Multiplatforms

Multi

By default, you are not allowed to specify a deprecated CipherSpec on a channel definition. If you attempt to specify a deprecated CipherSpec on IBM MQ for Multiplatforms, you receive message AMQ8242: SSLCIPH definition wrong, and PCF returns `MQRCCF_SSL_CIPHER_SPEC_ERROR`.

You cannot start a channel with a deprecated CipherSpec. If you attempt to do so with a deprecated CipherSpec, the system returns `MQCC_FAILED (2)`, together with a **Reason** of `MQRC_SSL_INITIALIZATION_ERROR (2393)` to the client.

You can re-enable one or more of the deprecated CipherSpecs for defining channels, at runtime on the server, by setting the environment variable **`AMQ_SSL_WEAK_CIPHER_ENABLE`**.

The **`AMQ_SSL_WEAK_CIPHER_ENABLE`** environment variable accepts:

- A single CipherSpec name, or
- A comma separated list of CipherSpec names to re-enable, or

- The special value of ALL, representing all CipherSpecs.

Attention: Although ALL is a valid option, you should use it **only** in a specific situation that your enterprise requires, as re-enabling ALL CipherSpecs enables SSL 3.0 and TLS 1.0 protocols, as well as a large number of weak cryptographic algorithms.

For example, if you want to re-enable ECDHE_RSA_RC4_128_SHA256, set the following environment variable:

```
  export AMQ_SSL_WEAK_CIPHER_ENABLE=ECDHE_RSA_RC4_128_SHA256
```

or, alternatively change the SSL stanza in the `qm.ini` file, by setting:

```
SSL:
  AllowTLSV1=Y
  AllowWeakCipherSpec=ECDHE_RSA_RC4_128_SHA256
```

Enabling deprecated CipherSpecs on z/OS

z/OS

By default, you are not allowed to specify a deprecated CipherSpec on a channel definition. If you attempt to specify a deprecated CipherSpec on z/OS, you receive message CSQM102E, message CSQX616E, or CSQX674E.

Follow the instructions listed in this section if you receive any one of these messages, and your enterprise needs to re-enable the use of weak CipherSpecs.

Attention: In the following instructions, for the dummy definition (DD) statements to take effect, SSLTASKS must be a non-zero value. If this requires a change to SSLTASKS you must recycle the channel initiator.

On IBM MQ for z/OS, the current method of controlling weak or broken CipherSpecs is as follows:

- If you want to re-enable the use of weak CipherSpecs, you do so by adding a dummy data definition (DD) statement named CSQXWEAK to the channel initiator JCL. If specified on its own, this only enables weak CipherSpecs associated with the TLS 1.2 protocol; for example:

  ```
  //CSQXWEAK DD DUMMY
  ```

  Note: Not all deprecated CipherSpecs require the use of this DD statement, see note 10 in the preceding table.
- If you want to re-enable the use of SSLv3 CipherSpecs, you do so by also adding a dummy DD statement named CSQXSSL3 to the channel initiator JCL. All SSLv3 CipherSpecs are considered **Weak**, so you must also specify CSQXWEAK:

  ```
  //CSQXSSL3 DD DUMMY
  ```
- If you want to re-enable the deprecated TLS V1 CipherSpecs, you do so by adding a dummy DD statement named TLS10ON (turn TLS V1.0 ON) to the channel initiator JCL. If specified on its own, this enables Strong CipherSpecs associated with the TLS 1.0 protocol:

  ```
  //TLS10ON DD DUMMY
  ```

  If specified with CSQXWEAK this also enables **Weak** CipherSpecs associated with TLS 1.0.
- If you want to explicitly turn off the deprecated TLS V1 CipherSpecs, you do so by adding a dummy DD statement named TLS10OFF (turn TLS V1.0 OFF) to the channel initiator JCL; for example:

  ```
  //TLS10OFF DD DUMMY
  ```

If you want to only negotiate with the listener using the cipher specifications listed on the **`System SSL`** default cipher specification list, you need to define the following DD statement in the CHINIT JCL:

```
JCL: //GSKDCIPS DD DUMMY
```

Important: For IBM MQ for z/OS 9.2.0 and later, the previously listed DD cards and the value of **`AllowTLSV13`** are taken into account when displaying messages during channel initiator startup to indicate which protocols are enabled and which are not. So, even if one of the previously listed DD cards is specified, it could mean that, due to a combination of these settings, a certain protocol cannot be enabled with another protocol. For example, protocol SSL 3.0 is not allowed if TLS 1.3 is enabled.

There are alternative mechanisms that can be used to forcibly re-enable weak CipherSpecs, and SSLv3 support, if the Data Definition change is unsuitable. Contact IBM Service for further information.

Related concepts
"Digital certificates and CipherSpec compatibility in IBM MQ" on page 47
This topic provides information on how to choose appropriate CipherSpecs and digital certificates for your security policy, by outlining the relationship between CipherSpecs and digital certificates in IBM MQ.

Related reference
DEFINE CHANNEL
ALTER CHANNEL

Relationship between alias CipherSpec settings

This information describes the expected behavior with different combinations of alias CipherSpecs in client and server configurations. Here, a client refers to the entity initiating communication, for example a client application or a queue manager sender channel, and server refers to the entity receiving the communication from the client, for example a server-connection channel or a receiver channel.

Minimum protocol versus fixed protocol CipherSpecs

IBM MQ supports two different types of CipherSpecs:

Minimum protocol
Minimum protocol CipherSpecs are those that do not set an upper bound, for example ANY, ANY_TLS12_OR_HIGHER or ANY_TLS13_OR_HIGHER.

Fixed protocol
Fixed protocol CipherSpecs are those that identify a specific protocol, for example ANY_TLS12 and ANY_TLS13, or a specific algorithm such as ECDHE_ECDSA_3DES_EDE_CBC_SHA256.

Minimum and fixed protocol CipherSpecs are supported on all platforms.

To maximize simplicity of configuration whilst maintaining security, the use of **minimum protocol** CipherSpecs is recommended on both sides of the channel. This allows your communications to automatically support and use a higher TLS protocol version when both sides support a new version without the need for changing either side's configuration.

Using a **minimum protocol** CipherSpec on the initiating side, but a **fixed protocol** CipherSpec on the receiving side could result in the connection being rejected, and

- Multi Messages AMQ9631 and AMQ9641 being issued.
- z/OS Messages CSQX631E and CSQX641E being issued.

The following tables show the relationship between different alias CipherSpec settings and the expected outcome. Table 81 on page 431 shows the expected behavior when TLS 1.3 is not enabled on either the client, server, or both. Table 82 on page 431 shows the expected behavior when TLS 1.3 is enabled on both the client and server. In both cases, the CipherSpecs for the client are shown in the Y axis of the table, and the CipherSpecs for the server are shown in the X axis of the table.

Note: In the following tables, cells marked *Likely to fail* indicate the potential for conflict when you specify a **minimum protocol** CipherSpec for one part of a connection, and a specific (**fixed protocol**) CipherSpec for another part.

For example, suppose the client and server are set to use ANY CipherSpec, and the server channel is set to use a specific CipherSpec:

- If the strongest supported CipherSpec for both the client and server matches the specific CipherSpec configured on the channel, the TLS handshake resolves successfully.
- If, however, there is a stronger CipherSpec that both the client and Server support, the TLS handshake resolves to using this, even though it does not match the CipherSpec specified on the channel, and the TLS handshake fails.

Table 81. Expected behavior when TLS 1.3 is not enabled on either the client, server, or both

	Server			
Client	**Specific TLS 1.2 CipherSpec**	**ANY**	**ANY_ TLS12**	**ANY_TLS12_ OR_HIGHER**
Specific TLS 1.2 CipherSpec	Connects	Connects	Connects	Connects
ANY	*Likely to fail*	Connects	Connects	Connects
ANY_ TLS12	*Likely to fail*	Connects	Connects	Connects
ANY_TLS12_ OR_HIGHER	*Likely to fail*	Connects	Connects	Connects

Table 82. Expected behavior when TLS 1.3 is enabled on both the client and server

	Server						
Client	**Specific TLS 1.2 CipherSpec**	**Specific TLS 1.3 CipherSpec**	**ANY**	**ANY_TLS 12**	**ANY_TLS 13**	**ANY_TLS12_ OR_HIGHER**	**ANY_TLS13_ OR_HIGHER**
Specific TLS 1.2 CipherSpec	Connects	**Fails**	Connects	Connects	**Fails**	Connects	**Fails**
Specific TLS 1.3 CipherSpec	**Fails**	Connects	Connects	**Fails**	Connects	Connects	Connects
ANY	**Fails**	*Likely to fail*	Connects	**Fails**	Connects	Connects	Connects
ANY_TLS12	*Likely to fail*	**Fails**	Connects	Connects	**Fails**	Connects	**Fails**
ANY_TLS13	**Fails**	*Likely to fail*	Connects	**Fails**	Connects	Connects	Connects
ANY_TLS12_ OR_HIGHER	**Fails**	*Likely to fail*	Connects	**Fails**	Connects	Connects	Connects
ANY_TLS13_ OR_HIGHER	**Fails**	*Likely to fail*	Connects	**Fails**	Connects	Connects	Connects

Related concepts

"Digital certificates and CipherSpec compatibility in IBM MQ" on page 47

This topic provides information on how to choose appropriate CipherSpecs and digital certificates for your security policy, by outlining the relationship between CipherSpecs and digital certificates in IBM MQ.

"CipherSpecs and CipherSuites" on page 22
Cryptographic security protocols must agree on the algorithms used by a secure connection. CipherSpecs and CipherSuites define specific combinations of algorithms.

"Enabling CipherSpecs" on page 411
Enable a CipherSpec by using the **SSLCIPH** parameter in either the **DEFINE CHANNEL** or **ALTER CHANNEL** MQSC command.

Related tasks

Migrating existing security configurations to use the ANY_TLS12_OR_HIGHER CipherSpec

Obtaining information about CipherSpecs using IBM MQ Explorer

You can use IBM MQ Explorer to display descriptions of CipherSpecs.

Use the following procedure to obtain information about the CipherSpecs in "Enabling CipherSpecs" on page 411:

1. Open IBM MQ Explorer and expand the **Queue Managers** folder.
2. Ensure that you have started your queue manager.
3. Select the queue manager you want to work with and click **Channels**.
4. Right-click the channel you want to work with and select **Properties**.
5. Select the **SSL** property page.
6. Select from the list the CipherSpec you want to work with. A description is displayed in the window below the list.

z/OS IBM i Alternatives for specifying CipherSpecs

For those platforms where the operating system provides the TLS support, your system might support new CipherSpecs that are not included in "Enabling CipherSpecs" on page 411.

You can specify a new CipherSpec with the SSLCIPH parameter, but the value you supply depends on your platform. In all cases the specification must correspond to an TLS CipherSpec that is both valid and supported by the version of TLS your system is running.

Note: This section does not apply to AIX, Linux, and Windows systems, because the CipherSpecs are provided with the IBM MQ product, so new CipherSpecs do not become available after shipment.

IBM i **IBM i**

A two-character string representing a hexadecimal value.

For more information about the permitted values, see point three in the Usage Notes section of Set character information for a secure session.

Attention: You should not specify hexadecimal cipher values in **SSLCIPH**, because it is unclear from the value which cipher will be used, and the choice of which protocol to be used is indeterminate. Using hexadecimal cipher values can lead to CipherSpec mismatch errors.

You can use either the **CHGMQMCHL** or the **CRTMQMCHL** command to specify the value, for example:

```
CRTMQMCHL CHLNAME(' channel name ') SSLCIPH(' hexadecimal value ')
```

You can also use the **ALTER QMGR** MQSC command to set the **SSLCIPH** parameter.

z/OS **z/OS**

A four-character string representing a hexadecimal value. The hexadecimal codes correspond to the values defined in the TLS protocol.

For more information, refer to Cipher Suite Definitions where there is a list of all the supported TLS 1.0, TLS 1.2, and TLS 1.3 cipher specifications in the form of 4-digit hexadecimal codes.

Note: Deprecated In order to use a weak CipherSpec, or a CipherSpec belonging to a deprecated protocol, such as SSL V3.0 or TLS 1.0, you must specify the relevant DD card in the channel initiator startup JCL. See "Deprecated CipherSpecs" on page 425 for more information.

Considerations for IBM MQ clusters

With IBM MQ clusters it is safest to use the CipherSpec names in "Enabling CipherSpecs" on page 411. If you use an alternative specification, be aware that the specification might not be valid on other platforms. For more information, refer to "SSL/TLS and clusters" on page 470.

Specifying a CipherSpec for an IBM MQ MQI client

You have three options for specifying a CipherSpec for an IBM MQ MQI client.

These options are as follows:

- Using a channel definition table
- Using the SSLCipherSpec field in the MQCD structure, at MQCD_VERSION_7 or higher, on an MQCONNX call.
- Using the Active Directory (on Windows systems with Active Directory support)

Specifying a CipherSuite with IBM MQ classes for Java and IBM MQ classes for JMS

IBM MQ classes for Java and IBM MQ classes for JMS specify CipherSuites differently from other platforms.

For information about specifying a CipherSuite with IBM MQ classes for Java, see Transport Layer Security (TLS) support for Java

For information about specifying a CipherSuite with IBM MQ classes for JMS, see Using Transport Layer Security (TLS) with IBM MQ classes for JMS

Specifying a CipherSpec for IBM MQ.NET

For IBM MQ.NET you can specify the CipherSpec either by using the MQEnvironment class or by using the MQC.SSL_CIPHER_SPEC_PROPERTY in the hash table of connection properties.

For information about specifying a CipherSpec for the .NET unmanaged client, see Enabling TLS for the unmanaged .NET client

For information about specifying a CipherSpec for the .NET managed client, see CipherSpec support for the managed .NET client

z/OS Use of AT-TLS with IBM MQ for z/OS

Application Transparent Transport Layer Security (AT-TLS) provides TLS support for z/OS applications without those applications having to implement TLS support, or even be aware that TLS is being used. AT-TLS is only available on z/OS.

AT-TLS can be used with all versions of IBM MQ for z/OS.

Before making use of AT-TLS with IBM MQ for z/OS, make sure you understand the "Restrictions" on page 437 involved.

To use Application Transparent Transport Layer Security you define policy statements containing a set of rules which are used by z/OS Communications Server to decide which TCP/IP connections have TLS transparently enabled.

IBM MQ for z/OS has its own TLS implementation, which requires that channels have the SSLCIPH parameter configured with a supported CipherSpec.

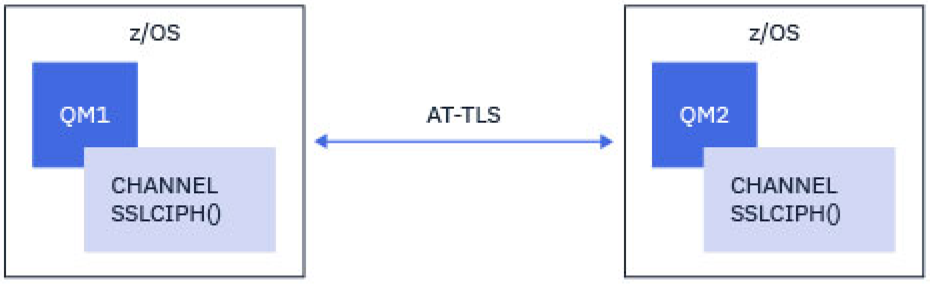
When deciding to enable TLS on a channel the IBM MQ administrator can decide to either use AT-TLS or IBM MQ TLS. The decision is often made based on whether AT-TLS is used for other middleware, or because of performance implications. For a basic comparison of the performance of AT-TLS and IBM MQ TLS see MP16: Capacity Planning and Tuning for IBM MQ for z/OS.

Scenarios

Use of AT-TLS with IBM MQ is supported in the following scenarios:

Scenario 1

Between two IBM MQ for z/OS queue managers where both sides of the channel use AT-TLS. That is, neither channel specifies the SSLCIPH attribute. This approach can be used with any message channel.



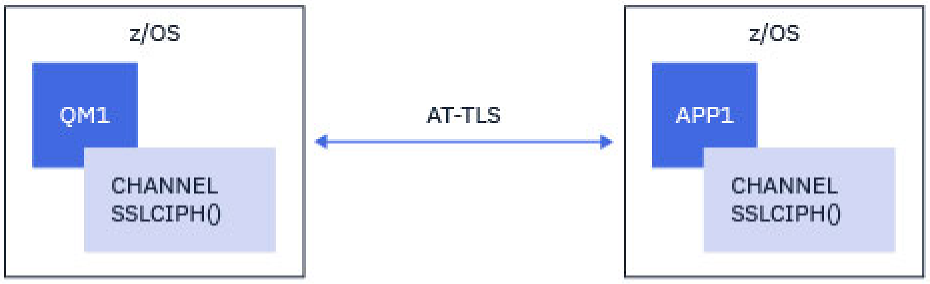
Implementation of this scenario consists of defining two AT-TLS policies, one for each side of the channel. These policies are the same as those used with either Scenario 3 or Scenario 4.

For example, if the channel was being changed from using a single, named CipherSpec to using AT-TLS, the outbound channel would use the policy from "Configuring AT-TLS on an outbound channel to an IBM MQ for Multiplatforms queue manager using a single, named CipherSpec" on page 438 and the inbound channel would use the policy from "Configuring AT-TLS on an inbound channel from an IBM MQ for Multiplatforms queue manager using a single, named CipherSpec" on page 446.

If the channel was being changed from using an alias CipherSpec to using AT-TLS, the outbound channel would use the policy from "Configuring AT-TLS on an outbound channel to an IBM MQ for Multiplatforms queue manager using alias CipherSpecs" on page 442 and the inbound channel would use the policy from "Configuring AT-TLS on an inbound channel from an IBM MQ for Multiplatforms queue manager using an alias CipherSpec" on page 451.

Scenario 2

Between an IBM MQ for z/OS queue manager and an IBM MQ Java client application running on z/OS where both sides of the channel use AT-TLS. That is, neither the server-connection channel, nor the client-connection channel specify the SSLCIPH attribute.



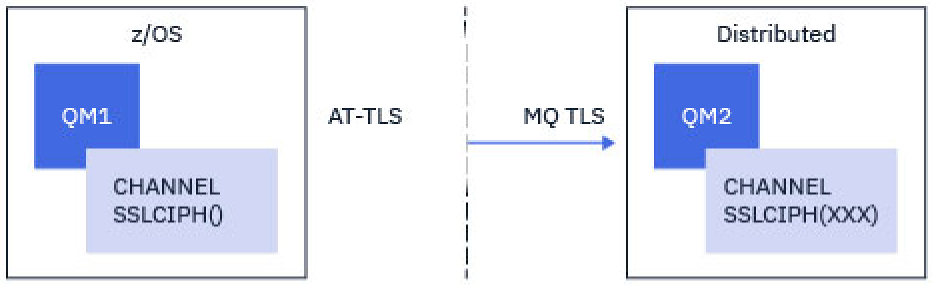
Implementation of this scenario consists of defining two AT-TLS policies, one for each side of the channel. These policies are the same as those used with either Scenario 3 or Scenario 4.

For example, if the channel was being changed from using a single, named CipherSpec to using AT-TLS the client-connection channel would use the policy from "Configuring AT-TLS on an outbound channel to an IBM MQ for Multiplatforms queue manager using a single, named CipherSpec" on page 438 and the server-connection channel would use the policy from "Configuring AT-TLS on an inbound channel from an IBM MQ for Multiplatforms queue manager using a single, named CipherSpec" on page 446.

If the channel was being changed from using an alias CipherSpec to using AT-TLS the client-connection channel would use the policy from "Configuring AT-TLS on an outbound channel to an IBM MQ for Multiplatforms queue manager using alias CipherSpecs" on page 442 and the server-connection channel would use the policy from "Configuring AT-TLS on an inbound channel from an IBM MQ for Multiplatforms queue manager using an alias CipherSpec" on page 451.

Scenario 3

Between an IBM MQ for z/OS queue manager and a queue manager running on IBM MQ for Multiplatforms, where the IBM MQ for z/OS queue manager uses AT-TLS and the IBM MQ for Multiplatforms queue manager uses IBM MQ TLS, by specifying the SSLCIPH attribute with a single named CipherSpec. This applies to all message channel types other than cluster-sender and cluster-receiver.

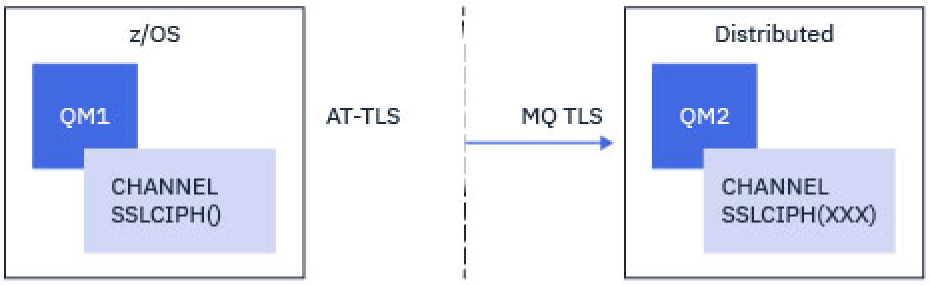


See "Configuring AT-TLS on an outbound channel to an IBM MQ for Multiplatforms queue manager using a single, named CipherSpec" on page 438 for an example AT-TLS configuration for outbound channels from the IBM MQ for z/OS queue manager to the IBM MQ for Multiplatforms queue manager, and "Configuring AT-TLS on an inbound channel from an IBM MQ for Multiplatforms queue manager using a single, named CipherSpec" on page 446 for an example AT-TLS configuration for inbound channels from the IBM MQ for Multiplatforms queue manager to the IBM MQ for z/OS queue manager.

The same AT-TLS configuration can be used when both queue managers are on z/OS, but the queue manager on the right hand side has not been configured to use AT-TLS.

Scenario 4

Between an IBM MQ for z/OS queue manager and a queue manager running on IBM MQ for Multiplatforms, where the IBM MQ for z/OS queue manager uses AT-TLS and the IBM MQ for Multiplatforms queue manager uses IBM MQ TLS, by specifying the SSLCIPH attribute with an alias CipherSpec. This applies to all message channel types other than cluster-sender and cluster-receiver.

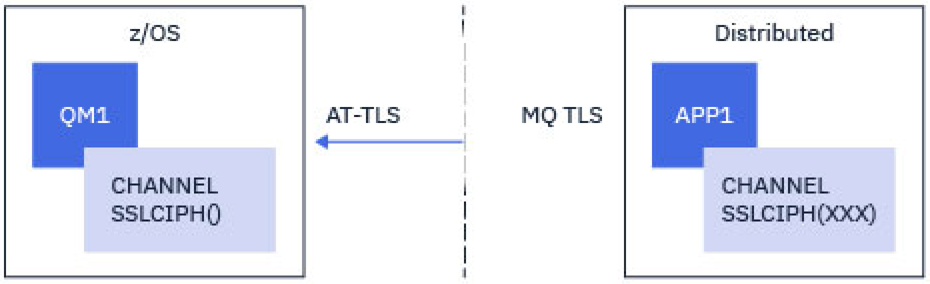


See “Configuring AT-TLS on an outbound channel to an IBM MQ for Multiplatforms queue manager using alias CipherSpecs” on page 442 for an example AT-TLS configuration for outbound channels from the IBM MQ for z/OS queue manager to the IBM MQ for Multiplatforms queue manager, and “Configuring AT-TLS on an inbound channel from an IBM MQ for Multiplatforms queue manager using an alias CipherSpec” on page 451, and “Configuring AT-TLS on an inbound channel from an IBM MQ for Multiplatforms queue manager using an alias CipherSpec” on page 451 for an example AT-TLS configuration for inbound channels from the IBM MQ for Multiplatforms queue manager to the IBM MQ for z/OS queue manager.

The same AT-TLS configuration can be used when both queue managers are on z/OS, but the queue manager on the right hand side has not been configured to use AT-TLS.

Scenario 5

Between an IBM MQ for z/OS queue manager and a client application running on IBM MQ for Multiplatforms, where the IBM MQ for z/OS queue manager uses AT-TLS and the client application uses IBM MQ TLS by specifying the SSLCIPH attribute with a single, named CipherSpec.

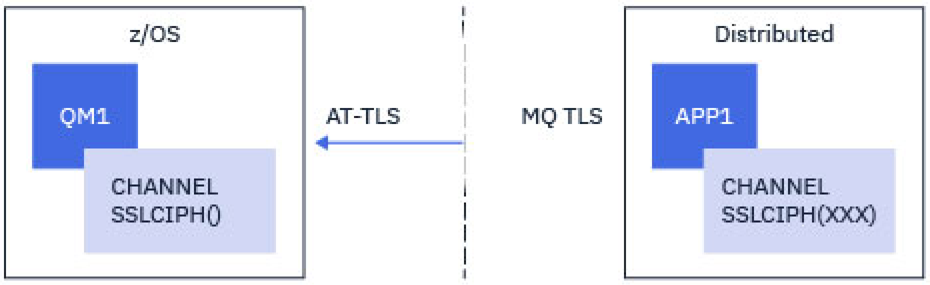


This scenario requires a single AT-TLS policy which meets the same requirements as those used by an inbound message channel; see “Configuring AT-TLS on an inbound channel from an IBM MQ for Multiplatforms queue manager using a single, named CipherSpec” on page 446.

The same AT-TLS configuration can be used when the client application is a Java application, and is also running on z/OS, but has not been configured to use AT-TLS.

Scenario 6

Between an IBM MQ for z/OS queue manager and a client application running on IBM MQ for Multiplatforms, where the IBM MQ for z/OS queue manager uses AT-TLS and the client application uses IBM MQ TLS by specifying the SSLCIPH attribute with an alias CipherSpec.



This scenario requires a single AT-TLS policy which meets the same requirements as those used by an inbound message channel; see "Configuring AT-TLS on an inbound channel from an IBM MQ for Multiplatforms queue manager using an alias CipherSpec" on page 451.

The same AT-TLS configuration can be used when the client application is a Java application, and is also running on z/OS, but has not been configured to use AT-TLS.

Restrictions

IBM MQ for z/OS is not AT-TLS aware, therefore there are several restrictions that apply with the preceding scenarios:

- AT-TLS in combination with IBM MQ TLS does not work with cluster-sender and cluster-receiver channels.
- IBM MQ for z/OS queue managers are not aware that they are using AT-TLS and do not receive any certificate information from their partner queue manager or client. Therefore, the following attributes have no effect on the z/OS side of a channel using AT-TLS:
  - The SSLCAUTH, and SSLPEER channel attributes
  - The SSLRKEYC queue manager attribute
  - The SSLPEERMAP attributes of CHLAUTH rules
- Use of TLS secret key renegotiation requires that both sides of the channel use IBM MQ TLS. Therefore, an IBM MQ for Multiplatforms queue manager, or client, should not have TLS secret key renegotiation enabled if connecting to an IBM MQ for z/OS queue manager using AT-TLS.

  To disable TLS secret key renegotiation for a queue manager set the queue manager SSLRKEYC parameter to `0`. For a client, set the relevant parameter to `0` depending on client type. For details on how to do this, see "Resetting SSL and TLS secret keys" on page 455.

AT-TLS configuration statements

AT-TLS is configured using a set of statements. The ones used in the scenarios documented in this topic are:

TTLSRule
: Specifies a set of criteria for matching a TCP/IP connection to a TLS configuration. This in turn refers to the other statement types.

TTLSGroupAction
: Specifies whether the referencing `TTLSRule` is enabled or not.

TTLSEnvironmentAction
: Specifies the detailed configuration for the referencing `TTLSRule` and references a number of other statements.

TTLSKeyringParms
: References the key-ring that is to be used by AT-TLS.

TTLSCipherParms
: Defines the cipher suites that are to be used.

TTLSEnvironmentAdvancedParms
Defines which TLS or SSL protocols are enabled.

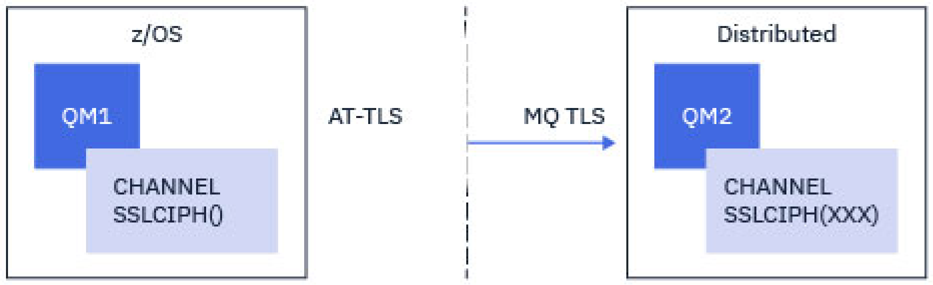
Attention: There are other AT-TLS policy statements with AT-TLS which are not documented here, and could be used with IBM MQ depending on need. However, IBM MQ has only been tested with the policies described in this topic.

z/OS *Configuring AT-TLS on an outbound channel to an IBM MQ for Multiplatforms queue manager using a single, named CipherSpec*

How you set up AT-TLS on an outbound channel from an IBM MQ for z/OS queue manager to an IBM MQ for Multiplatforms queue manager. In this case, the channel on the z/OS queue manager is a sender channel which does not have the SSLCIPH attribute set, and the channel on the non-z/OS queue manager is a receiver channel with the SSLCIPH attribute set to a single, named CipherSpec.

See "Configuring AT-TLS on an outbound channel to an IBM MQ for Multiplatforms queue manager using alias CipherSpecs" on page 442 for an example using an alias CipherSpec.

In this example an existing sender – receiver channel pair, which uses the TLS 1.3 TLS_AES_256_GCM_SHA384 CipherSpec is going to be adjusted so that the sender channel uses AT-TLS instead of IBM MQ TLS.



Other TLS protocols and CipherSpecs can be used by making minor adjustments to the configuration. Other message channel types, apart from cluster-sender and cluster-receiver channels, could be used with no change to the AT-TLS configuration.

Procedure

Step 1: Stop the channel

Step 2: Create and apply an AT-TLS policy

You need to create the following AT-TLS statements for this scenario:

1. A TTLSRule statement to match outbound connections from the channel initiator address space to the IP address and port number of the target receiver channel. These values should match the information used in the CONNAME of the sender channel. Here, further filtering has been included to match a specific channel initiator job name.

```
TTLSRule                      CSQ1-TO-REMOTE
{
  LocalAddr                   ALL
  RemoteAddr                  123.456.78.9
  RemotePortRange             1414
  Jobname                     CSQ1CHIN
  Direction                   OUTBOUND
  TTLSGroupActionRef          CSQ1-GROUP-ACTION
  TTLSEnvironmentActionRef    CSQ1-OUTBOUND-ENVIRONMENT-ACTION
}
```

The preceding rule matches against connections going to IP address 123.456.78.9 on port 1414 from the CSQ1CHIN job.

More advanced filtering options are described at TTLSRule.

2. A TTLSGroupAction statement enabling the rule. The TTLSRule references the TTLSGroupAction using the **TTLSGroupActionRef** property.

```
TTLSGroupAction              CSQ1-GROUP-ACTION
{
  TTLSEnabled                ON
}
```

3. A TTLSEnvironmentAction statement associated with the TTLSRule by the **TTLSEnvironmentActionRef** property. A TTLSEnvironmentAction configures the TLS Environment and specifies which key ring to use.

```
TTLSEnvironmentAction                CSQ1-OUTBOUND-ENVIRONMENT-ACTION
{
  HandshakeRole                      CLIENT
  TTLSKeyringParmsRef                CSQ1-KEYRING
  TTLSCipherParmsRef                 CSQ1-CIPHERPARM
  TTLSEnvironmentAdvancedParmsRef    CSQ1-ENVIRONMENT-ADVANCED
}
```

4. A TTLSKeyringParms statement associated with the TTLSEnvironmentAction by the **TTLSKeyringParmsRef** property and defines the key ring used by AT-TLS.

   The key ring should contain certificates trusted by the remote non-z/OS queue manager. This key ring can be defined in the same way as a key ring used by the channel initiator; see "Configuring your z/OS system to use TLS" on page 250.

```
TTLSKeyringParms             CSQ1-KEYRING
{
  Keyring                    MQCHIN/CSQ1RING
}
```

5. A TTLSCipherParms statement associated with the TTLSEnvironmentAction by the **TTLSCipherParmsRef** property.

   This statement must contain a single cipher suite name which must be the equivalent of the IBM MQ CipherSpec name used on the target receiver channel.

   Note: AT-TLS cipher suite names do not necessarily match IBM MQ CipherSpec names. However, it is possible to find the AT-TLS cipher suite name that matches an IBM MQ CipherSpec name by finding the IBM MQ CipherSpec name in the following table and cross-referencing the hexadecimal code column with the expanded character column from Table 2 in the TTLSCipherParms statement topic.

Table 83. CipherSpecs on z/OS from IBM MQ for z/OS 9.2.0

CipherSpec	Protocol	Hexadecimal code	Enabled by default
TLS_CHACHA20_POLY1305_SHA256	TLS 1.3	1303	Yes
TLS_AES_256_GCM_SHA384	TLS 1.3	1302	Yes
TLS_AES_128_GCM_SHA256	TLS 1.3	1301	Yes
TLS_RSA_WITH_AES_256_GCM_SHA384	TLS 1.2	009D	Yes

Table 83. CipherSpecs on z/OS from IBM MQ for z/OS 9.2.0 (continued)

CipherSpec	Protocol	Hexadecimal code	Enabled by default
ECDHE_RSA_AES_256_GCM_SHA384	TLS 1.2	C030	Yes
TLS_RSA_WITH_AES_256_CBC_SHA256	TLS 1.2	003D	Yes
ECDHE_ECDSA_AES_256_CBC_SHA384	TLS 1.2	C024	Yes
ECDHE_RSA_AES_256_CBC_SHA384	TLS 1.2	C028	Yes
TLS_RSA_WITH_AES_128_GCM_SHA256	TLS 1.2	009C	Yes
ECDHE_RSA_AES_128_GCM_SHA256	TLS 1.2	C02F	Yes
TLS_RSA_WITH_AES_128_CBC_SHA256	TLS 1.2	003C	Yes
ECDHE_ECDSA_AES_128_CBC_SHA256	TLS 1.2	C023	Yes
ECDHE_RSA_AES_128_CBC_SHA256	TLS 1.2	C027	Yes
TLS_RSA_WITH_NULL_SHA256	TLS 1.2	003B	No
TLS_RSA_WITH_AES_256_CBC_SHA	TLS 1.0	0035	No
TLS_RSA_WITH_AES_128_CBC_SHA	TLS 1.0	002F	No
TLS_RSA_WITH_3DES_EDE_CBC_SHA	TLS 1.0	000A	No
TLS_RSA_WITH_RC4_128_SHA	TLS 1.0	0005	No
TLS_RSA_WITH_DES_CBC_SHA	TLS 1.0	0009	No
TRIPLE_DES_SHA_US	SSL v3	000A	No
RC4_SHA_US	SSL v3	0005	No
RC4_MD5_US	SSL v3	0004	No
DES_SHA_EXPORT	SSL v3	0009	N
RC4_MD5_EXPORT	SSL v3	0003	No
RC2_MD5_EXPORT	SSL v3	0006	No
NULL_SHA	SSL v3	0002	No
NULL_MD5	SSL v3	0001	No

```
TTLSCipherParms             CSQ1-CIPHERPARM
{
  V3CipherSuites            TLS_AES_256_GCM_SHA384
}
```

6. A TTLSEnvironmentAdvancedParms statement is associated with the `TTLSEnvironmentAction` by the **`TTLSEnvironmentAdvancedParmsRef`** property.

   This statement can be used to specify which SSL and TLS protocols are enabled. With IBM MQ you should enable only the single protocol that matches the cipher suite name used on the `TTLSCipherParms` statement.

```
TTLSEnvironmentAdvancedParms CSQ1-ENVIRONMENT-ADVANCED
{
  SSLv3          OFF
  TLSv1          OFF
  TLSv1.1        OFF
  SecondaryMap   OFF
  TLSv1.2        OFF
  TLSv1.3        ON
}
```

The complete set of statements are as follows and should be applied to the policy agent:

```
TTLSRule                      CSQ1-TO-REMOTE
{
  LocalAddr                   ALL
  RemoteAddr                  123.456.78.9
  RemotePortRange             1414
  Jobname                     CSQ1CHIN
  Direction                   OUTBOUND
  TTLSGroupActionRef          CSQ1-GROUP-ACTION
  TTLSEnvironmentActionRef    CSQ1-OUTBOUND-ENVIRONMENT-ACTION
}

TTLSGroupAction             CSQ1-GROUP-ACTION
{
  TTLSEnabled               ON
}

TTLSEnvironmentAction             CSQ1-OUTBOUND-ENVIRONMENT-ACTION
{
  HandshakeRole                   CLIENT
  TTLSKeyringParmsRef             CSQ1-KEYRING
  TTLSCipherParmsRef              CSQ1-CIPHERPARM
  TTLSEnvironmentAdvancedParmsRef CSQ1-ENVIRONMENT-ADVANCED
}

TTLSKeyringParms            CSQ1-KEYRING
{
  Keyring                   MQCHIN/CSQ1RING
}

TTLSCipherParms             CSQ1-CIPHERPARM
{
  V3CipherSuites            TLS_AES_256_GCM_SHA384
}

TTLSEnvironmentAdvancedParms CSQ1-ENVIRONMENT-ADVANCED
{
  SSLv3          OFF
  TLSv1          OFF
  TLSv1.1        OFF
  SecondaryMap   OFF
  TLSv1.2        OFF
  TLSv1.3        ON
}
```

Step 3: Remove SSLCIPH from the z/OS channel

Remove the CipherSpec from the z/OS channel using the following command:

```
ALTER CHANNEL(channel-name) CHLTYPE(SDR) SSLCIPH(' ')
```

Step 4: Start the channel

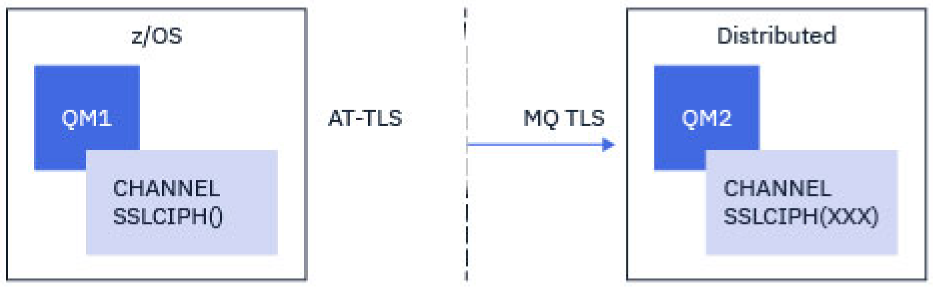
Once the channel has started it will be using a combination of AT-TLS and IBM MQ TLS.

Attention: The preceding AT-TLS statements are only a minimal configuration. There are other AT-TLS policy statements with AT-TLS which are not documented here, and could be used with IBM MQ depending on need. However, IBM MQ has only been tested with the policies described.

z/OS *Configuring AT-TLS on an outbound channel to an IBM MQ for Multiplatforms queue manager using alias CipherSpecs*

How you set up AT-TLS on an outbound channel from an IBM MQ for z/OS queue manager to an IBM MQ for Multiplatforms queue manager. In this case, the channel on the z/OS queue manager is a sender channel which does not have the SSLCIPH attribute set, and the channel on the non-z/OS queue manager is a receiver channel with the SSLCIPH attribute set to an alias CipherSpec

In this example an existing sender – receiver channel pair, which uses the ANY_TLS13 alias CipherSpec is going to be adjusted so that the sender channel uses AT-TLS instead of IBM MQ TLS.



Other TLS protocols and CiperSpecs can be used by making minor adjustments to the configuration. Other message channel types, apart from cluster-sender and cluster-receiver channels, could be used with no change to the AT-TLS configuration.

Procedure

Step 1: Stop the channel

Step 2: Create and apply an AT-TLS policy

You need to create the following AT-TLS statements for this scenario:

1. A TTLSRule statement to match outbound connections from the channel initiator address space to the IP address and port number of the target receiver channel. These values should match the information used in the CONNAME of the sender channel. Here, further filtering has been included to match a specific channel initiator job name.

```
TTLSRule                      CSQ1-TO-REMOTE
{
  LocalAddr                   ALL
  RemoteAddr                  123.456.78.9
  RemotePortRange             1414
  Jobname                     CSQ1CHIN
  Direction                   OUTBOUND
  TTLSGroupActionRef          CSQ1-GROUP-ACTION
  TTLSEnvironmentActionRef    CSQ1-OUTBOUND-ENVIRONMENT-ACTION
}
```

The preceding rule matches against connections going to IP address 123.456.78.9 on port 1414 from the CSQ1CHIN job.

More advanced filtering options are described at TTLSRule.

2. A TTLSGroupAction statement enabling the rule. The TTLSRule references the TTLSGroupAction using the **TTLSGroupActionRef** property.

```
TTLSGroupAction              CSQ1-GROUP-ACTION
{
  TTLSEnabled                ON
}
```

3. A TTLSEnvironmentAction statement associated with the TTLSRule by the **TTLSEnvironmentActionRef** property. A TTLSEnvironmentAction configures the TLS Environment and specifies which key ring to use.

```
TTLSEnvironmentAction                CSQ1-OUTBOUND-ENVIRONMENT-ACTION
{
  HandshakeRole                      CLIENT
  TTLSKeyringParmsRef                CSQ1-KEYRING
  TTLSCipherParmsRef                 CSQ1-CIPHERPARM
  TTLSEnvironmentAdvancedParmsRef    CSQ1-ENVIRONMENT-ADVANCED
}
```

4. A TTLSKeyringParms statement associated with the TTLSEnvironmentAction by the **TTLSKeyringParmsRef** property and defines the key ring used by AT-TLS.

   The key ring should contain certificates trusted by the remote non-z/OS queue manager. This key ring can be defined in the same way as a key ring used by the channel initiator; see "Configuring your z/OS system to use TLS" on page 250.

```
TTLSKeyringParms             CSQ1-KEYRING
{
  Keyring                    MQCHIN/CSQ1RING
}
```

5. A TTLSCipherParms statement associated with the TTLSEnvironmentAction by the **TTLSCipherParmsRef** property.

   This statement must contain one or more cipher suite names, at least one of which should be compatible with the set of CipherSpecs implied by the alias CipherSpec used on the target receiver channel.

   Note: AT-TLS cipher suite names do not necessarily match IBM MQ CipherSpec names. However, it is possible to find the AT-TLS cipher suite name that matches an IBM MQ CipherSpec name by finding the IBM MQ CipherSpec name in the following table and cross-referencing the hexadecimal code column with the expanded character column from Table 2 in the TTLSCipherParms topic.

Table 84. CipherSpecs on z/OS from IBM MQ for z/OS 9.2.0

CipherSpec	Protocol	Hexadecimal code	Enabled by default
TLS_CHACHA20_POLY1305_SHA256	TLS 1.3	1303	Yes
TLS_AES_256_GCM_SHA384	TLS 1.3	1302	Yes
TLS_AES_128_GCM_SHA256	TLS 1.3	1301	Yes
TLS_RSA_WITH_AES_256_GCM_SHA384	TLS 1.2	009D	Yes

Table 84. CipherSpecs on z/OS from IBM MQ for z/OS 9.2.0 (continued)			
CipherSpec	**Protocol**	**Hexadecimal code**	**Enabled by default**
ECDHE_RSA_AES_256_GCM_SHA384	TLS 1.2	C030	Yes
TLS_RSA_WITH_AES_256_CBC_SHA256	TLS 1.2	003D	Yes
ECDHE_ECDSA_AES_256_CBC_SHA384	TLS 1.2	C024	Yes
ECDHE_RSA_AES_256_CBC_SHA384	TLS 1.2	C028	Yes
TLS_RSA_WITH_AES_128_GCM_SHA256	TLS 1.2	009C	Yes
ECDHE_RSA_AES_128_GCM_SHA256	TLS 1.2	C02F	Yes
TLS_RSA_WITH_AES_128_CBC_SHA256	TLS 1.2	003C	Yes
ECDHE_ECDSA_AES_128_CBC_SHA256	TLS 1.2	C023	Yes
ECDHE_RSA_AES_128_CBC_SHA256	TLS 1.2	C027	Yes
TLS_RSA_WITH_NULL_SHA256	TLS 1.2	003B	No
TLS_RSA_WITH_AES_256_CBC_SHA	TLS 1.0	0035	No
TLS_RSA_WITH_AES_128_CBC_SHA	TLS 1.0	002F	No
TLS_RSA_WITH_3DES_EDE_CBC_SHA	TLS 1.0	000A	No
TLS_RSA_WITH_RC4_128_SHA	TLS 1.0	0005	No
TLS_RSA_WITH_DES_CBC_SHA	TLS 1.0	0009	No
TRIPLE_DES_SHA_US	SSL v3	000A	No
RC4_SHA_US	SSL v3	0005	No
RC4_MD5_US	SSL v3	0004	No
DES_SHA_EXPORT	SSL v3	0009	N
RC4_MD5_EXPORT	SSL v3	0003	No
RC2_MD5_EXPORT	SSL v3	0006	No
NULL_SHA	SSL v3	0002	No
NULL_MD5	SSL v3	0001	No

```
TTLSCipherParms              CSQ1-CIPHERPARM
{
  V3CipherSuites             TLS_CHACHA20_POLY1305_SHA256
  V3CipherSuites             TLS_AES_256_GCM_SHA384
  V3CipherSuites             TLS_AES_128_GCM_SHA256
}
```

Attention: If both the queue manager and AT-TLS policy support TLS 1.3, only alias CipherSpecs that contain at least one TLS 1.3 CipherSpec allow the channel to start. For example, using ANY_TLS12 results in the channel failing to start, even if TTLSCipherParms contains TLS 1.2 CipherSpecs, but using ANY_TLS12_OR_HIGHER or ANY_TLS13 allows the channel to start. See "Relationship between alias CipherSpec settings" on page 430 for an explanation.

6. A TTLSEnvironmentAdvancedParms statement is associated with the `TTLSEnvironmentAction` by the **`TTLSEnvironmentAdvancedParmsRef`** property.

   This statement can be used to specify which SSL and TLS protocols are enabled, and should be consistent with the cipher suites in the `TTLSCipherParms` statement.

```
TTLSEnvironmentAdvancedParms CSQ1-ENVIRONMENT-ADVANCED
{
  SSLv3          OFF
  TLSv1          OFF
  TLSv1.1        OFF
  SecondaryMap   OFF
  TLSv1.2        OFF
  TLSv1.3        ON
}
```

The complete set of statements are as follows and should be applied to the policy agent :

```
TTLSRule                        CSQ1-TO-REMOTE
{
  LocalAddr                     ALL
  RemoteAddr                    123.456.78.9
  RemotePortRange               1414
  Jobname                       CSQ1CHIN
  Direction                     OUTBOUND
  TTLSGroupActionRef            CSQ1-GROUP-ACTION
  TTLSEnvironmentActionRef      CSQ1-OUTBOUND-ENVIRONMENT-ACTION
}

TTLSGroupAction               CSQ1-GROUP-ACTION
{
  TTLSEnabled                 ON
}

TTLSEnvironmentAction               CSQ1-OUTBOUND-ENVIRONMENT-ACTION
{
  HandshakeRole                     CLIENT
  TTLSKeyringParmsRef               CSQ1-KEYRING
  TTLSCipherParmsRef                CSQ1-CIPHERPARM
  TTLSEnvironmentAdvancedParmsRef CSQ1-ENVIRONMENT-ADVANCED
}

TTLSKeyringParms              CSQ1-KEYRING
{
  Keyring                     MQCHIN/CSQ1RING
}

TTLSCipherParms               CSQ1-CIPHERPARM
{
  V3CipherSuites              TLS_CHACHA20_POLY1305_SHA256
  V3CipherSuites              TLS_AES_256_GCM_SHA384
  V3CipherSuites              TLS_AES_128_GCM_SHA256

}

TTLSEnvironmentAdvancedParms CSQ1-ENVIRONMENT-ADVANCED
{
  SSLv3          OFF
  TLSv1          OFF
  TLSv1.1        OFF
  SecondaryMap   OFF
  TLSv1.2        OFF
  TLSv1.3        ON
}
```

Step 3: Remove SSLCIPH from the z/OS channel

Remove the CipherSpec from the z/OS channel using the following command:

```
ALTER CHANNEL(channel-name) CHLTYPE(SDR) SSLCIPH(' ')
```

Step 4: Start the channel

Once the channel has started it will be using a combination of AT-TLS and IBM MQ TLS.

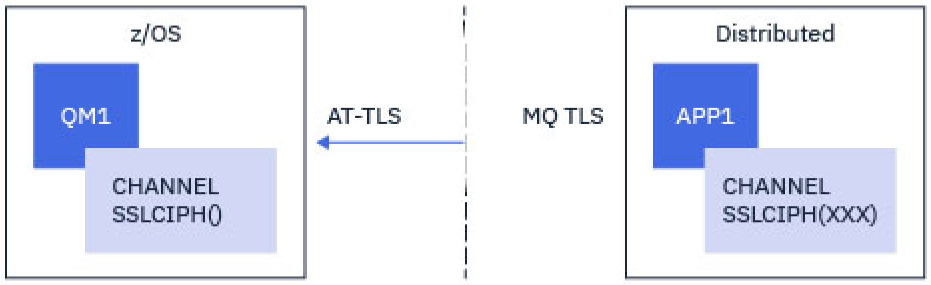
Attention: The preceding AT-TLS statements are only a minimal configuration. There are other AT-TLS policy statements with AT-TLS which are not documented here, and could be used with IBM MQ depending on need. However, IBM MQ has only been tested with the policies described.

z/OS *Configuring AT-TLS on an inbound channel from an IBM MQ for Multiplatforms queue manager using a single, named CipherSpec*

How you set up AT-TLS on an inbound channel from an IBM MQ for Multiplatforms queue manager to an IBM MQ for z/OS queue manager. In this case, the channel on the z/OS queue manager is a receiver channel which does not have the SSLCIPH attribute set, and the channel on the non-z/OS queue manager is a sender channel with the SSLCIPH attribute set to a single, named CipherSpec.

See "Configuring AT-TLS on an inbound channel from an IBM MQ for Multiplatforms queue manager using an alias CipherSpec" on page 451 for an example using an alias CipherSpec.

In this example an existing sender – receiver channel pair, which uses the TLS 1.3 TLS_AES_256_GCM_SHA384 CipherSpec is going to be adjusted so that the receiver channel uses AT-TLS instead of IBM MQ TLS.



Other TLS protocols and CipherSpecs can be used by making minor adjustments to the configuration. Other message channel types, apart from cluster-sender and cluster-receiver channels, could be used with no change to the AT-TLS configuration.

Procedure

Step 1: Stop the channel

Step 2: Create and apply an AT-TLS policy

You need to create the following AT-TLS statements for this scenario:

1. A TTLSRule statement to match inbound connections to the channel initiator address space from the IP address of the sender channel. Here, further filtering has been included to match a specific channel initiator job name.

```
TTLSRule                      REMOTE-TO-CSQ1
{
  LocalAddr                   ALL
  LocalPortRange              1414
  RemoteAddr                  123.456.78.9
  Jobname                     CSQ1CHIN
  Direction                   INBOUND
  TTLSGroupActionRef          CSQ1-GROUP-ACTION
  TTLSEnvironmentActionRef    CSQ1-INBOUND-ENVIRONMENT-ACTION
}
```

   The preceding rule matches against connections coming into the CSQ1CHIN job on local port 1414 from remote IP address 123.456.78.9.

   More advanced filtering options are described at TTLSRule.

2. A TTLSGroupAction statement enabling the rule. The `TTLSRule` references the `TTLSGroupAction` using the **`TTLSGroupActionRef`** property.

```
TTLSGroupAction               CSQ1-GROUP-ACTION
{
  TTLSEnabled                 ON
}
```

3. A TTLSEnvironmentAction statement is associated with the `TTLSRule` by the **`TTLSEnvironmentActionRef`** property. A `TTLSEnvironmentAction` configures the TLS Environment and specifies which key ring to use.

```
TTLSEnvironmentAction                CSQ1-INBOUND-ENVIRONMENT-ACTION
{
  HandshakeRole                      SERVER
  TTLSKeyringParmsRef                CSQ1-KEYRING
  TTLSCipherParmsRef                 CSQ1-CIPHERPARM
  TTLSEnvironmentAdvancedParmsRef    CSQ1-ENVIRONMENT-ADVANCED
}
```

AT-TLS provides the capability to provide mutual authentication, which is the equivalent of using the SSLCAUTH channel attribute. This is done by having an `TTLSEnvironmentAction` statement with a **`HandshakeRole`** value of *ServerWithClientAuth* for the inbound `TTLSEnvironmentAction` statement.

4. A TTLSKeyringParms statement is associated with the `TTLSEnvironmentAction` by the **`TTLSKeyringParmsRef`** property and defines the key ring used by AT-TLS.

   The key ring should contain certificates trusted by the remote non-z/OS queue manager. This key ring can be defined in the same way as a key ring used by the channel initiator; see "Configuring your z/OS system to use TLS" on page 250.

```
TTLSKeyringParms            CSQ1-KEYRING
{
  Keyring                   MQCHIN/CSQ1RING
}
```

5. A TTLSCipherParms statement associated with the `TTLSEnvironmentAction` by the **`TTLSCipherParmsRef`** property.

   This statement must contain a single cipher suite name which must be the equivalent of the IBM MQ CipherSpec name used on the remote sender channel.

   Note: AT-TLS cipher suite names do not necessarily match IBM MQ CipherSpec names. However, it is possible to find the AT-TLS cipher suite name that matches an IBM MQ CipherSpec name by finding the IBM MQ CipherSpec name in the following table and cross-referencing the hexadecimal code column with the expanded character column from Table 2 in the TTLSCipherParms statement topic.

Table 85. CipherSpecs on z/OS from IBM MQ for z/OS 9.2.0

CipherSpec	Protocol	Hexadecimal code	Enabled by default
TLS_CHACHA20_POLY1305_SHA256	TLS 1.3	1303	Yes
TLS_AES_256_GCM_SHA384	TLS 1.3	1302	Yes
TLS_AES_128_GCM_SHA256	TLS 1.3	1301	Yes
TLS_RSA_WITH_AES_256_GCM_SHA384	TLS 1.2	009D	Yes
ECDHE_RSA_AES_256_GCM_SHA384	TLS 1.2	C030	Yes
TLS_RSA_WITH_AES_256_CBC_SHA256	TLS 1.2	003D	Yes
ECDHE_ECDSA_AES_256_CBC_SHA384	TLS 1.2	C024	Yes
ECDHE_RSA_AES_256_CBC_SHA384	TLS 1.2	C028	Yes

Table 85. CipherSpecs on z/OS from IBM MQ for z/OS 9.2.0 (continued)

CipherSpec	Protocol	Hexadecimal code	Enabled by default
TLS_RSA_WITH_AES_128_GCM_SHA256	TLS 1.2	009C	Yes
ECDHE_RSA_AES_128_GCM_SHA256	TLS 1.2	C02F	Yes
TLS_RSA_WITH_AES_128_CBC_SHA256	TLS 1.2	003C	Yes
ECDHE_ECDSA_AES_128_CBC_SHA256	TLS 1.2	C023	Yes
ECDHE_RSA_AES_128_CBC_SHA256	TLS 1.2	C027	Yes
TLS_RSA_WITH_NULL_SHA256	TLS 1.2	003B	No
TLS_RSA_WITH_AES_256_CBC_SHA	TLS 1.0	0035	No
TLS_RSA_WITH_AES_128_CBC_SHA	TLS 1.0	002F	No
TLS_RSA_WITH_3DES_EDE_CBC_SHA	TLS 1.0	000A	No
TLS_RSA_WITH_RC4_128_SHA	TLS 1.0	0005	No
TLS_RSA_WITH_DES_CBC_SHA	TLS 1.0	0009	No
TRIPLE_DES_SHA_US	SSL v3	000A	No
RC4_SHA_US	SSL v3	0005	No
RC4_MD5_US	SSL v3	0004	No
DES_SHA_EXPORT	SSL v3	0009	N
RC4_MD5_EXPORT	SSL v3	0003	No
RC2_MD5_EXPORT	SSL v3	0006	No
NULL_SHA	SSL v3	0002	No
NULL_MD5	SSL v3	0001	No

```
TTLSCipherParms              CSQ1-CIPHERPARM
{
  V3CipherSuites             TLS_AES_256_GCM_SHA384
}
```

6. A TTLSEnvironmentAdvancedParms statement is associated with the `TTLSEnvironmentAction` by the **`TTLSEnvironmentAdvancedParmsRef`** property.

   This statement can be used to specify which SSL and TLS protocols are enabled. With IBM MQ you should enable only the single protocol that matches the cipher suite name used on the `TTLSCipherParms` statement.

```
TTLSEnvironmentAdvancedParms CSQ1-ENVIRONMENT-ADVANCED
{
  SSLv3          OFF
  TLSv1          OFF
  TLSv1.1        OFF
  SecondaryMap   OFF
  TLSv1.2        OFF
  TLSv1.3        ON
}
```

The complete set of statements are as follows and should be applied to the policy agent :

```

TTLSRule                          REMOTE-TO-CSQ1
{
  LocalAddr                       ALL
  LocalPortRange                  1414
  RemoteAddr                      123.456.78.9
  Jobname                         CSQ1CHIN
  Direction                       INBOUND
  TTLSGroupActionRef              CSQ1-GROUP-ACTION
  TTLSEnvironmentActionRef        CSQ1-INBOUND-ENVIRONMENT-ACTION
}

TTLSGroupAction                 CSQ1-GROUP-ACTION
{
  TTLSEnabled                   ON
}

TTLSEnvironmentAction                 CSQ1-INBOUND-ENVIRONMENT-ACTION
{
  HandshakeRole                       SERVER
  TTLSKeyringParmsRef                 CSQ1-KEYRING
  TTLSCipherParmsRef                  CSQ1-CIPHERPARM
  TTLSEnvironmentAdvancedParmsRef CSQ1-ENVIRONMENT-ADVANCED
}

TTLSKeyringParms                CSQ1-KEYRING
{
  Keyring                       MQCHIN/CSQ1RING
}

TTLSCipherParms                 CSQ1-CIPHERPARM
{
  V3CipherSuites                TLS_AES_256_GCM_SHA384
}

TTLSEnvironmentAdvancedParms CSQ1-ENVIRONMENT-ADVANCED
{
  SSLv3          OFF
  TLSv1          OFF
  TLSv1.1        OFF
  SecondaryMap   OFF
  TLSv1.2        OFF
  TLSv1.3        ON
}
```

Step 3: Remove SSLCIPH from the z/OS channel

Remove the CipherSpec from the z/OS channel using the following command:

```
ALTER CHANNEL(channel-name) CHLTYPE(RCVR) SSLCIPH(' ')
```

Step 4: Start the channel

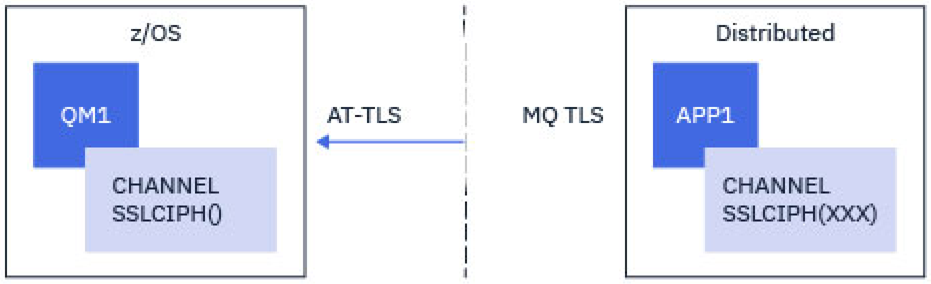
Once the channel has started it will be using a combination of AT-TLS and IBM MQ TLS.

Attention: The preceding AT-TLS statements are only a minimal configuration. There are other AT-TLS policy statements with AT-TLS which are not documented here, and could be used with IBM MQ depending on need. However, IBM MQ has only been tested with the policies described.

z/OS *Configuring AT-TLS on an inbound channel from an IBM MQ for Multiplatforms queue manager using an alias CipherSpec*

How you set up AT-TLS on an inbound channel from an IBM MQ for Multiplatforms queue manager to an IBM MQ for z/OS queue manager. In this case, the channel on the z/OS queue manager is a receiver channel which does not have the SSLCIPH attribute set, and the channel on the non-z/OS queue manager is a sender channel with the SSLCIPH attribute set to an alias CipherSpec.

In this example an existing sender – receiver channel pair, which uses any TLS 1.3 CipherSpec is going to be adjusted so that the receiver channel uses AT-TLS instead of IBM MQ TLS.



Other TLS protocols and CiperSpecs can be used by making minor adjustments to the configuration. Other message channel types, apart from cluster-sender and cluster-receiver channels, could be used with no change to the AT-TLS configuration.

Procedure

Step 1: Stop the channel

Step 2: Create and apply an AT-TLS policy

You need to create the following AT-TLS statements for this scenario:

1. A TTLSRule statement to match inbound connections to the channel initiator address space from the IP address of the sender channel. Here, further filtering has been included to match a specific channel initiator job name.

```
TTLSRule                      REMOTE-TO-CSQ1
{
  LocalAddr                   ALL
  LocalPortRange              1414
  RemoteAddr                  123.456.78.9
  Jobname                     CSQ1CHIN
  Direction                   INBOUND
  TTLSGroupActionRef          CSQ1-GROUP-ACTION
  TTLSEnvironmentActionRef    CSQ1-INBOUND-ENVIRONMENT-ACTION
}
```

   The preceding rule matches against connections coming into the CSQ1CHIN job on local port 1414 from remote IP address 123.456.78.9.

   More advanced filtering options are described at TTLSRule.

2. A TTLSGroupAction statement enabling the rule. The `TTLSRule` references the `TTLSGroupAction` using the **`TTLSGroupActionRef`** property.

```
TTLSGroupAction             CSQ1-GROUP-ACTION
{
  TTLSEnabled               ON
}
```

3. A TTLSEnvironmentAction statement is associated with the TTLSRule by the **TTLSEnvironmentActionRef** property. A TTLSEnvironmentAction configures the TLS Environment and specifies which key ring to use.

```
TTLSEnvironmentAction                  CSQ1-INBOUND-ENVIRONMENT-ACTION
{
  HandshakeRole                        SERVER
  TTLSKeyringParmsRef                  CSQ1-KEYRING
  TTLSCipherParmsRef                   CSQ1-CIPHERPARM
  TTLSEnvironmentAdvancedParmsRef      CSQ1-ENVIRONMENT-ADVANCED
}
```

   AT-TLS provides the capability to provide mutual authentication, which is the equivalent of using the SSLCAUTH channel attribute. This is done by having an TTLSEnvironmentAction statement with a **HandshakeRole** value of *ServerWithClientAuth* for the inbound TTLSEnvironmentAction statement.

4. A TTLSKeyringParms statement is associated with the TTLSEnvironmentAction by the **TTLSKeyringParmsRef** property and defines the key ring used by AT-TLS.

   The key ring should contain certificates trusted by the remote non-z/OS queue manager. This key ring can be defined in the same way as a key ring used by the channel initiator; see "Configuring your z/OS system to use TLS" on page 250.

```
TTLSKeyringParms            CSQ1-KEYRING
{
  Keyring                   MQCHIN/CSQ1RING
}
```

5. A TTLSCipherParms statement associated with the TTLSEnvironmentAction by the **TTLSCipherParmsRef** property.

   This statement must contain at least one cipher suite name which is included in the alias CipherSpec set on the remote sender channel.

   Note: AT-TLS cipher suite names do not necessarily match IBM MQ CipherSpec names. However, it is possible to find the AT-TLS cipher suite name that matches an IBM MQ CipherSpec name by finding the IBM MQ CipherSpec name in the following table and cross-referencing the hexadecimal code column with the expanded character column from Table 2 in the TTLSCipherParms statement topic.

Table 86. CipherSpecs on z/OS from IBM MQ for z/OS 9.2.0

CipherSpec	Protocol	Hexadecimal code	Enabled by default
TLS_CHACHA20_POLY1305_SHA256	TLS 1.3	1303	Yes
TLS_AES_256_GCM_SHA384	TLS 1.3	1302	Yes
TLS_AES_128_GCM_SHA256	TLS 1.3	1301	Yes
TLS_RSA_WITH_AES_256_GCM_SHA384	TLS 1.2	009D	Yes
ECDHE_RSA_AES_256_GCM_SHA384	TLS 1.2	C030	Yes
TLS_RSA_WITH_AES_256_CBC_SHA256	TLS 1.2	003D	Yes
ECDHE_ECDSA_AES_256_CBC_SHA384	TLS 1.2	C024	Yes

Table 86. CipherSpecs on z/OS from IBM MQ for z/OS 9.2.0 (continued)

CipherSpec	Protocol	Hexadecimal code	Enabled by default
ECDHE_RSA_AES_256_CBC_SHA384	TLS 1.2	C028	Yes
TLS_RSA_WITH_AES_128_GCM_SHA256	TLS 1.2	009C	Yes
ECDHE_RSA_AES_128_GCM_SHA256	TLS 1.2	C02F	Yes
TLS_RSA_WITH_AES_128_CBC_SHA256	TLS 1.2	003C	Yes
ECDHE_ECDSA_AES_128_CBC_SHA256	TLS 1.2	C023	Yes
ECDHE_RSA_AES_128_CBC_SHA256	TLS 1.2	C027	Yes
TLS_RSA_WITH_NULL_SHA256	TLS 1.2	003B	No
TLS_RSA_WITH_AES_256_CBC_SHA	TLS 1.0	0035	No
TLS_RSA_WITH_AES_128_CBC_SHA	TLS 1.0	002F	No
TLS_RSA_WITH_3DES_EDE_CBC_SHA	TLS 1.0	000A	No
TLS_RSA_WITH_RC4_128_SHA	TLS 1.0	0005	No
TLS_RSA_WITH_DES_CBC_SHA	TLS 1.0	0009	No
TRIPLE_DES_SHA_US	SSL v3	000A	No
RC4_SHA_US	SSL v3	0005	No
RC4_MD5_US	SSL v3	0004	No
DES_SHA_EXPORT	SSL v3	0009	N
RC4_MD5_EXPORT	SSL v3	0003	No
RC2_MD5_EXPORT	SSL v3	0006	No
NULL_SHA	SSL v3	0002	No
NULL_MD5	SSL v3	0001	No

```
TTLSCipherParms                 CSQ1-CIPHERPARM
{
  V3CipherSuites                TLS_CHACHA20_POLY1305_SHA256
  V3CipherSuites                TLS_AES_256_GCM_SHA384
  V3CipherSuites                TLS_AES_128_GCM_SHA256
}
```

Attention: If both the queue manager and AT-TLS policy support TLS 1.3, only alias CipherSpecs that contain at least one TLS 1.3 CipherSpec allow the channel to start. For example, using ANY_TLS12 results in the channel failing to start, even if TTLSCipherParms contains TLS 1.2 CipherSpecs, but using ANY_TLS12_OR_HIGHER or ANY_TLS13 allows the channel to start. See "Relationship between alias CipherSpec settings" on page 430 for an explanation.

6. A TTLSEnvironmentAdvancedParms statement is associated with the TTLSEnvironmentAction by the **TTLSEnvironmentAdvancedParmsRef** property.

   This statement can be used to specify which SSL and TLS protocols are enabled, and should be consistent with the cipher suites in the TTLSCipherParms statement.

```
TTLSEnvironmentAdvancedParms CSQ1-ENVIRONMENT-ADVANCED
{
  SSLv3          OFF
  TLSv1          OFF
  TLSv1.1        OFF
  SecondaryMap   OFF
  TLSv1.2        OFF
  TLSv1.3        ON
}
```

The complete set of statements are as follows and should be applied to the policy agent :

```

TTLSRule                          REMOTE-TO-CSQ1
{
  LocalAddr                       ALL
  LocalPortRange                  1414
  RemoteAddr                      123.456.78.9
  Jobname                         CSQ1CHIN
  Direction                       INBOUND
  TTLSGroupActionRef              CSQ1-GROUP-ACTION
  TTLSEnvironmentActionRef        CSQ1-INBOUND-ENVIRONMENT-ACTION
}

TTLSGroupAction                 CSQ1-GROUP-ACTION
{
  TTLSEnabled                   ON
}

TTLSEnvironmentAction                 CSQ1-INBOUND-ENVIRONMENT-ACTION
{
  HandshakeRole                       SERVER
  TTLSKeyringParmsRef                 CSQ1-KEYRING
  TTLSCipherParmsRef                  CSQ1-CIPHERPARM
  TTLSEnvironmentAdvancedParmsRef CSQ1-ENVIRONMENT-ADVANCED
}

TTLSKeyringParms                CSQ1-KEYRING
{
  Keyring                       MQCHIN/CSQ1RING
}

TTLSCipherParms                 CSQ1-CIPHERPARM
{
  V3CipherSuites                TLS_CHACHA20_POLY1305_SHA256
  V3CipherSuites                TLS_AES_256_GCM_SHA384
  V3CipherSuites                TLS_AES_128_GCM_SHA256

}

TTLSEnvironmentAdvancedParms CSQ1-ENVIRONMENT-ADVANCED
{
  SSLv3          OFF
  TLSv1          OFF
  TLSv1.1        OFF
  SecondaryMap   OFF
  TLSv1.2        OFF
  TLSv1.3        ON
}
```

Step 3: Remove SSLCIPH from the z/OS channel

Remove the CipherSpec from the z/OS channel using the following command:

```
ALTER CHANNEL(channel-name) CHLTYPE(SDR) SSLCIPH(' ')
```

Step 4: Start the channel

Once the channel has started it will be using a combination of AT-TLS and IBM MQ TLS.

Attention: The preceding AT-TLS statements are only a minimal configuration. There are other AT-TLS policy statements with AT-TLS which are not documented here, and could be used with IBM MQ depending on need. However, IBM MQ has only been tested with the policies described.

Resetting SSL and TLS secret keys

IBM MQ supports the resetting of secret keys on queue managers and clients.

Secret keys are reset when a specified number of encrypted bytes of data have flowed across the channel. If channel heartbeats are enabled, the secret key is reset before data is sent or received following a channel heartbeat.

The key reset value is always set by the initiating side of the IBM MQ channel.

Queue manager

For a queue manager, use the command **`ALTER QMGR`** with the parameter **SSLRKEYC** to set the values used during key renegotiation.

IBM i On IBM i, use **`CHGMQM`** with the **SSLRSTCNT** parameter.

MQI client

By default, MQI clients do not renegotiate the secret key. You can make an MQI client renegotiate the key in any of three ways. In the following list, the methods are shown in order of priority. If you specify multiple values, the highest priority value is used.

1. By using the KeyResetCount field in the MQSCO structure on an MQCONNX call.
2. By using the environment variable **`MQSSLRESET`**.
3. By setting the **`SSLKeyResetCount`** attribute in the SSL stanza of the client configuration file.

These variables can be set to an integer in the range 0 through 999 999 999, representing the number of unencrypted bytes sent and received within a TLS conversation before the TLS secret key is renegotiated. Specifying a value of 0 indicates that TLS secret keys are never renegotiated. If you specify a TLS secret key reset count in the range 1 byte through 32 KB, TLS channels will use a secret key reset count of 32 KB. This is to avoid excessive key resets which would occur for small TLS secret key reset values.

If a value greater than zero is specified and channel heartbeats are enabled for the channel, the secret key is also renegotiated before message data is sent or received following a channel heartbeat.

The count of bytes until the next secret key renegotiation is reset after each successful renegotiation.

Java

For IBM MQ classes for Java, an application can reset the secret key in either of the following ways:

- By setting the sslResetCount field in the MQEnvironment class.
- By setting the environment property MQC.SSL_RESET_COUNT_PROPERTY in a Hashtable object. The application then assigns the hashtable to the `properties` field in the MQEnvironment class, or passes the hashtable to an MQQueueManager object on its constructor.

If the application uses more than one of these ways, the usual precedence rules apply. See Class com.ibm.mq.MQEnvironment for the precedence rules.

The value of the sslResetCount field or environment property MQC.SSL_RESET_COUNT_PROPERTY represents the total number of bytes sent and received by the IBM MQ classes for Java client code before the secret key is renegotiated. The number of bytes sent is the number before encryption, and the number of bytes received is the number after decryption. The number of bytes also includes control information sent and received by the IBM MQ classes for Java client.

If the reset count is zero, which is the default value, the secret key is never renegotiated. The reset count is ignored if no CipherSuite is specified.

JMS

For IBM MQ classes for JMS, the SSLRESETCOUNT property represents the total number of bytes sent and received by a connection before the secret key that is used for encryption is renegotiated. The number of bytes sent is the number before encryption, and the number of bytes received is the number after decryption. The number of bytes also includes control information sent and received by IBM MQ classes for JMS. For example, to configure a ConnectionFactory object that can be used to create a connection over a TLS enabled MQI channel with a secret key that is renegotiated after 4 MB of data have flowed, issue the following command to JMSAdmin:

```
ALTER CF(my.cf) SSLRESETCOUNT(4194304)
```

If the value of SSLRESETCOUNT is zero, which is the default value, the secret key is never renegotiated. The SSLRESETCOUNT property is ignored if SSLCIPHERSUITE is not set.

.NET

For .NET unmanaged clients, the integer property **`SSLKeyResetCount`** indicates the number of unencrypted bytes sent and received within a TLS conversation before the secret key is renegotiated. For more information about the use of object properties in IBM MQ classes for .NET, see Getting and setting attribute values.

For .NET managed clients, the SSLStream class does not support secret key reset/renegotiation. However, to be consistent with other IBM MQ clients, the IBM MQ managed .NET client allows applications to set **`SSLKeyResetCount`**. For more information, see Secret key reset or renegotiation.

XMS .NET

For XMS .NET unmanaged clients, see Secure connections to an IBM MQ queue manager.

Related reference
ALTER QMGR
DISPLAYQMGR
Change Message Queue Manager (CHGMQM)
Display Message Queue Manager (DSPMQM)

Implementing confidentiality in user exit programs

Implementing confidentiality in security exits

Security exits can play a role in the confidentiality service by generating and distributing the symmetric key for encrypting and decrypting the data that flows on the channel. A common technique for doing this uses PKI technology.

One security exit generates a random data value, encrypts it with the public key of the queue manager or user that the partner security exit is representing, and sends the encrypted data to its partner in a security message. The partner security exit decrypts the random data value with the private key of the queue manager or user it is representing. Each security exit can now use the random data value to derive the symmetric key independently of the other by using an algorithm known to both of them. Alternatively, they can use the random data value as the key.

If the first security exit has not authenticated its partner by this time, the next security message sent by the partner can contain an expected value encrypted with the symmetric key. The first security exit can now authenticate its partner by checking that the partner security exit was able to encrypt the expected value correctly.

The security exits can also use this opportunity to agree the algorithm for encrypting and decrypting the data that flows on the channel, if more than one algorithm is available for use.

Implementing confidentiality in message exits

A message exit at the sending end of a channel can encrypt the application data in a message and another message exit at the receiving end of the channel can decrypt the data. For performance reasons, a symmetric key algorithm is normally used for this purpose. For more information about how the symmetric key can be generated and distributed, see "Implementing confidentiality in user exit programs" on page 456.

Headers in a message, such as the transmission queue header, MQXQH, which includes the embedded message descriptor, must not be encrypted by a message exit. This is because data conversion of the message headers takes place either after a message exit is called at the sending end or before a message exit is called at the receiving end. If the headers are encrypted, data conversion fails and the channel stops.

Implementing confidentiality in send and receive exits

Send and receive exits can be used to encrypt and decrypt the data that flows on a channel. They are more appropriate than message exits for providing this service for the following reasons:

- On a message channel, message headers can be encrypted as well as the application data in the messages.
- Send and receive exits can be used on MQI channels as well as message channels. Parameters on MQI calls might contain sensitive application data that needs to be protected while it flows on an MQI channel. You can therefore use the same send and receive exits on both kinds of channels.

Implementing confidentiality in the API exit and API-crossing exit

The application data in a message can be encrypted by an API or API-crossing exit when the message is put by the sending application and decrypted by a second exit when the message is retrieved by the receiving application. For performance reasons, a symmetric key algorithm is typically used for this purpose. However, at the application level, where many users might be sending messages to each other, the problem is how to ensure that only the intended receiver of a message is able to decrypt the message. One solution is to use a different symmetric key for each pair of users that send messages to each other. But this solution might be difficult and time consuming to administer, particularly if the users belong to different organizations. A standard way of solving this problem is known as *digital enveloping* and uses PKI technology.

When an application puts a message on a queue, an API or API-crossing exit generates a random symmetric key and uses the key to encrypt the application data in the message. The exit encrypts the symmetric key with the public key of the intended receiver. It then replaces the application data in the message with the encrypted application data and the encrypted symmetric key. In this way, only the intended receiver can decrypt the symmetric key and therefore the application data. If an encrypted message has more than one possible intended receiver, the exit can encrypt a copy of the symmetric key for each intended receiver.

If different algorithms for encrypting and decrypting the application data are available for use, the exit can include the name of the algorithm it has used.

z/OS Confidentiality for data at rest on IBM MQ for z/OS with data set encryption

IBM MQ for z/OS can harden customer and configuration data by writing the data to the active log data sets, the archive log data sets, page sets, boot strap data sets (BSDS), and shared message data sets (SMDS).

z/OS provides efficient, policy-based encryption of data sets. IBM MQ for z/OS supports z/OS data set encryption for:

- Active log data sets; see note "1" on page 458
- Archive log data sets; see note "2" on page 458
- Page sets; see note "1" on page 458
- BSDS; see note "2" on page 458
- CSQINP* data sets; see note "2" on page 458
- SMDS; see note "1" on page 458

This provides confidentiality of data at rest on an individual z/OS queue manager.

Notes:

1. From IBM MQ for z/OS 9.2.0, z/OS data set encryption for active logs. page sets, and SMDS are supported.
2. Data set encryption for archive logs, BSDS and CSQINP* data sets is supported on all versions of IBM MQ for z/OS.
3. IBM MQ Advanced Message Security provides an alternative mechanism of protecting data at rest. In addition AMS also protects data in memory and in flight

See Using the z/OS data set encryption enhancements for more information about z/OS data set encryption.

Configuration of z/OS data set encryption is outside of the control of IBM MQ for z/OS. Encryption settings take effect when the data set is created.

This means that any existing data sets need to be recreated before a new data set encryption policy can be used.

IBM MQ for z/OS can run with a mixture of encrypted and non-encrypted data sets, but a standard configuration would encrypt all, or none, of the data sets used.

z/OS Overview of steps to encrypt an IBM MQ for z/OS data set

How you encrypt an IBM MQ for z/OS data set.

Before you begin

You must ensure that you have configured z/OS data set encryption correctly in your enterprise. If you are setting up data set encryption in a queue sharing group, you must configure z/OS data set encryption for data sharing.

Note: A z/OS encrypted data set must be an extended format data set.

Procedure

1. Set up encryption key and `key-label` in RACF to use to encrypt the data set.
2. Create a profile for `key-label` in the RACF CSFKEYS class.
3. Grant READ access to the user Id of the queue manager, and any other user Ids that need access to the encrypted data.

This might include user IDs that are used to run print utilities against the data set. For example, the user running CSQUTIL SCOPY would need to decrypt the relevant page set.

4. Associate the encryption `key-label` with the data set name.

   You can do this by using an SMS data class, or a RACF DFP segment, for the data set name or high-level qualifier.

   You can also associate the `key-label` with the data set when the data set is allocated.
5. Rename any existing data set using IDCAMS ALTER.
6. Re-allocate the data set with the appropriate attributes.
7. Copy the contents of the renamed data set to the new data set using IDCAMS REPRO.

   The data is encrypted by the action of copying it into the data set.
8. Repeat steps "4" on page 459 to "6" on page 459 for any other data sets that need to be encrypted.

z/OS Example of how to encrypt queue manager active logs

The following topics guide you through the process of enabling data set encryption on existing active logs.

Note: The process for other data sets is similar to that for active logs.

In this example:

- Queue manager CSQ1 is run under user QMCSQ1, and has active log data sets CSQ1.LOGS.LOGCOPY1.DS001, CSQ1.LOGS.LOGCOPY1.DS002, and so on
- The hardware and software environment is capable of using z/OS data set encryption
- RACF is used as the SAF
- The queue manager has been stopped

Carry out the procedure in the following order:

1. "Configuring the data set encryption key for the queue manager" on page 459
2. "Configuring data set encryption for the log data sets" on page 460

z/OS Configuring the data set encryption key for the queue manager

How you configure a data set encryption key for a queue manager.

About this task

This task is a prerequisite for "Configuring data set encryption for the log data sets" on page 460.

Procedure

1. Set up an AES-256 bit encryption DATA key with a label, for example, CSQ1DSKY, using the z/OS key generator utility program (KGUP).
2. Define the RACF CSFKEYS profile for the CSQ1DSKY encryption key, by issuing the following command:

   ```
   RDEFINE CSFKEYS CSQ1DSKY UACC(NONE)
   ```
3. Configure the ICSF segment of the profile to allow the key to be used as a protected key, by issuing the following command:

   ```
   RALTER CSFKEYS CSQ1DSKY ICSF(SYMCPACFWRAP(YES) SYMCPACFRET(YES))
   ```
4. Allow the queue manager to use the encryption key by giving QMCSQ1 READ access to the profile, by issuing the following command:

   ```
   PERMIT CSQ1DSKY CLASS(CSFKEYS) ID(QMCSQ1) ACCESS(READ)
   ```

   Give the same access to any administrative user that needs to read or write the encrypted data set.
5. Refresh the CSFKEYS class by issuing the following command.

```
SETROPTS RACLIST(CSFKEYS) REFRESH
```

What to do next

Configure data set encryption for the data sets as described in "Configuring data set encryption for the log data sets" on page 460

z/OS Configuring data set encryption for the log data sets

How you configure the encryption on the log data sets.

Before you begin

Ensure that you have read:

Overview of steps to encrypt an IBM MQ for z/OS data set, and carried out the procedure in "Configuring the data set encryption key for the queue manager" on page 459

About this task

This method uses the DFP segment of a RACF generic profile, so that you can use the encryption key for all new data sets that match the profile.

Alternatively, you can configure and use an SMS data class, or the key label can be specified directly when allocating the data set.

As previously described, in this example, queue manager CSQ1 is run under user QMCSQ1, and has active log data sets CSQ1.LOGS.LOGCOPY1.DS001, CSQ1.LOGS.LOGCOPY1.DS002, and so on.

Procedure

1. Create the generic profile if it does not exist, by issuing the following command:

   ```
   ADDSD 'CSQ1.LOGS.*' UACC(NONE)
   ```

2. Permit the queue manager user alter access on the profile, by issuing the following command:

   ```
   PERMIT 'CSQ1.LOGS.*' ID(QMCSQ1) ACCESS(ALTER)
   ```

   Also, permit the appropriate access needed for any administrative user.

3. Add the DFP segment with the encryption key label by issuing the following command:

   ```
   ALTDSD 'CSQ1.LOGS.*' DFP(RESOWNER(QMCSQ1) DATAKEY(CSQ1DSKY))
   ```

   Note: You must use the same encryption key that you used in configuring the data set encryption key for the queue manager.

4. Refresh the generic dataset profiles by issuing the following command:

   ```
   SETROPTS GENERIC(DATASET) REFRESH
   ```

5. Rename each log data set to a backup, then recreate and restore the data, using IDCAMS. The following JCL fragment converts CSQ1.LOGS.LOGCOPY1.DS001:

   a) Rename the data set to a back-up

   ```
   //RENAME   EXEC PGM=IDCAMS,REGION=0M
   //SYSPRINT DD SYSOUT=*
   //SYSIN    DD *
     /*------------------------------------------------------------*/
     /* RENAME DATASET TO BACKUP                                   */
     /*------------------------------------------------------------*/
     ALTER 'CSQ1.LOGS.LOGCOPY1.DS001'                     -
           NEWNAME('CSQ1.BAK.LOGS.LOGCOPY1.DS001')
   ```

b) Redefine the data set.

The new data set will be encrypted due to the RACF profile.

Note: Replace ++EXTDCLASS++ with the name of the extended format data class you want to use for the data set.

```
//REDEFINE EXEC PGM=IDCAMS,REGION=0M
//SYSPRINT DD SYSOUT=*
//SYSIN    DD *
  /*-------------------------------------------------------------*/
  /* REDEFINE THE DATASET                                        */
  /*-------------------------------------------------------------*/
  DEFINE CLUSTER                                     -
         (NAME(CSQ1.LOGS.LOGCOPY1.DS001)             -
         LINEAR                                      -
         SHAREOPTIONS(2 3)                           -
         MODEL(CSQ1.BAK.LOGS.LOGCOPY1.DS001)         -
         DATACLAS(++EXTDCLASS++))
```

c) Copy the data from the backup into the recreated data set.

This step encrypts the data:

```
//RESTORE  EXEC PGM=IDCAMS,REGION=0M
//SYSPRINT DD SYSOUT=*
//SYSIN    DD *
  /*-------------------------------------------------------------*/
  /* RESTORE DATA INTO ENCRYPTED LOG                             */
  /*-------------------------------------------------------------*/
  REPRO INDATASET(CSQ1.BAK.LOGS.LOGCOPY1.DS001)      -
        OUTDATASET(CSQ1.LOGS.LOGCOPY1.DS001)
```

What to do next

Repeat Step "5" on page 460 for all active log data sets.

Only a single encryption key is required, and all data sets can be associated with the same key label.

Restart queue manager CSQ1. Use the output from the DISPLAY LOG command to verify that the log data sets have been encrypted.

z/OS Considerations for z/OS data set encryption in a queue sharing group

Each queue manager in a queue sharing group (QSG) must be able to read the logs, BSDS, and shared message data sets (SMDS), of every other queue manager in the QSG.

This means that each system on which a member of the QSG can run, must meet the requirements for z/OS data set encryption, and all the key labels and encryption keys used to protect the data sets for each queue manager in the QSG must be available on each system.

A queue manager prior to IBM MQ for z/OS 9.1.4 cannot access an encrypted active log data set.

A queue manager prior to IBM MQ for z/OS 9.1.5 cannot access an encrypted SMDS.

Before making use of z/OS data set encryption, you should migrate all queue managers in a QSG to at least IBM MQ for z/OS 9.1.5.

If a queue manager in a QSG is started with any encrypted active log data set, and any other queue manager in the QSG has been started, but was not last started with a version of IBM MQ for z/OS that supports encrypted active logs, the queue manager with the encrypted active log terminates abnormally with abend code 5C6-00F50033.

You can convert a QSG to use encrypted active logs and SMDS without a full outage, by:

1. Migrating each queue manager to at least IBM MQ for z/OS 9.1.5 in turn.

2. Converting active logs to encrypted data sets for each queue manager in turn. This requires the queue manager to be shut down and then restarted.

   At the same time, it is likely that page sets and archive logs would be enabled for encrypted data sets too, but this does not affect QSG migration.

   The procedure for converting each data set is described in "Example of how to encrypt queue manager active logs" on page 459
3. Converting SMDS to encrypted data sets for each individual CF structure in turn by:
   a. Issuing the command RESET SMDS(*) ACCESS(DISABLED) CFSTRUCT(structure-name) to suspend queue manager access to the SMDS.

      Note that during this time, the data on the shared queues associated with the SMDS is temporarily unavailable.
   b. Converting each data set that makes up the SMDS to encrypted data sets, using the procedure described in "Example of how to encrypt queue manager active logs" on page 459.
   c. Issuing the command RESET SMDS(*) ACCESS(ENABLED) CFSTRUCT(structure-name) to resume queue manager access to the SMDS.

Attention: You should shut the queue manager down cleanly prior to converting the logs, and coupling facility structure recovery might not be possible during the conversion, as the active log data sets will be temporarily unavailable.

z/OS Backwards migration considerations when using z/OS data set encryption

You need to consider the following when backwards migrating a queue manager, which has one or more encrypted data sets.

z/OS data set encryption is supported on the following IBM MQ for z/OS data sets:

- Active log data sets
- Archive log data sets
- Page sets
- BSDS
- SMDS
- CSQINP* data sets

There are no backwards migration considerations for BSDS, archive log, or CSINP* data sets.

However, there are considerations for

- SMDS
- Page set and
- Active log

data sets, as using these with z/OS data set encryption is not supported in IBM MQ for z/OS 9.1.0, and earlier, long term support releases.

Prior to backwards migration, all encryption policies for SMDS, page set, and active log data sets need to be removed and the data decrypted. This process is described in "Removing data set encryption from a data set" on page 463.

Attention: If the queue manager to be backwards migrated is part of a queue sharing group (QSG), read the "Queue sharing group considerations" on page 464 section first.

Removing data set encryption from a data set

This example describes how to remove data set encryption from the log data set CSQ1.LOGS.LOGCOPY1.DS001. You can use an equivalent process for SMDS and page sets.

The example assumes that:

- RACF is the SAF
- The queue manager that uses the data set has been stopped
- The encryption key label has been associated with the generic RACF profile CSQ1.LOGS.*

Carry out the following procedure:

1. Copy the data from the data set to a back-up data set.

   a. Define a backup data set which is not associated with an encryption key label.

      Note: Replace ++EXTDCLASS++ with the name of the extended format data class you want to use for the data set.

```
//DEFINE EXEC PGM=IDCAMS,REGION=0M
//SYSPRINT DD SYSOUT=*
//SYSIN    DD *
  /*-------------------------------------------------------------*/
  /* DEFINE UNENCRYPTED DATA SET                                 */
  /*-------------------------------------------------------------*/
  DEFINE CLUSTER                                      -
         (NAME(CSQ1.BAK.LOGS.LOGCOPY1.DS001)          -
         LINEAR                                       -
         SHAREOPTIONS(2 3)                            -
         MODEL(CSQ1.LOGS.LOGCOPY1.DS001)              -
         DATACLAS(++EXTDCLASS++))
/*
```

   b. Copy the data from the original data set to the backup. This step decrypts the data.

```
//COPY     EXEC PGM=IDCAMS,REGION=0M
//SYSPRINT DD SYSOUT=*
//SYSIN    DD *
  /*-------------------------------------------------------------*/
  /* COPY DATA INTO UNENCRYPTED DATA SET                         */
  /*-------------------------------------------------------------*/
  REPRO INDATASET(CSQ1.LOGS.LOGCOPY1.DS001)            -
        OUTDATASET(CSQ1.BAK.LOGS.LOGCOPY1.DS001)
/*
```

   c. Delete the original data set

```
//DELETE   EXEC PGM=IDCAMS,REGION=0M
//SYSPRINT DD SYSOUT=*
//SYSIN    DD *
  /*-------------------------------------------------------------*/
  /* DELETE ORIGINAL                                             */
  /*-------------------------------------------------------------*/
  DELETE ('CSQ1.LOGS.LOGCOPY1.DS001')
/*
```

   d. Rename the backup to the original data set name. The data remains unencrypted

```
//RENAME   EXEC PGM=IDCAMS,REGION=0M
//SYSPRINT DD SYSOUT=*
//SYSIN    DD *
  /*-------------------------------------------------------------*/
  /* RENAME UNENCRYPTED DATA SET                                 */
  /*-------------------------------------------------------------*/
  ALTER CSQ1.BAK.LOGS.LOGCOPY1.DS001'                  -
         NEWNAME('CSQ1.LOGS.LOGCOPY1.DS001)
  ALTER 'CSQ1.BAK.LOGS.LOGCOPY1.DS001.*'                -
         NEWNAME('CSQ1.LOGS.LOGCOPY1.DS001.*')
/*
```

2. Optionally, repeat this process for other data sets that have an encryption key label associated with them through the CSQ1.LOGS.* generic profile.
3. Optionally, if all data sets associated with the CSQ1.LOGS.* generic profile have been decrypted, remove the DATAKEY associated with the generic profile by issuing the following command

   ```
   ALTDSD 'CSQ1.LOGS.*' DFP(RESOWNER(QMCSQ1) DATAKEY(CSQ1DSKY))
   ```

4. Refresh the generic dataset profiles by issuing the following command:

   ```
   SETROPTS GENERIC(DATASET) REFRESH
   ```

5. Restart the queue manager.
6. If the encryption key is no longer needed, delete it, and delete its associated RACF profile from the CSFKEYS class.

Queue sharing group considerations

If a queue manager that is part of a queue sharing group is going to be backwards migrated to a version of IBM MQ for z/OS that does not support data set encryption then all of the active log data sets and SMDS of all queue managers in the QSG need to have their data set encryption policies removed, and their data decrypted.

This applies regardless of whether a single member of QSG is backwards migrated, or all members of the QSG.

You can achieve removal of encryption policies, and decryption of data, without a full QSG outage by:

1. Shutting down each queue manager in the QSG in turn, removing the encryption policies and decrypting the data from its active logs, using the process described in "Removing data set encryption from a data set" on page 463.

   If the queue manager is to be backwards migrated, its page set should also be decrypted at this time. Then restart the queue manager.
2. Removing the encryption policies and decrypting the data for the SMDS of each individual CF structure in turn by:

   a. Issuing the command

      ```
      RESET SMDS(*) ACCESS(DISABLED) CFSTRUCT(structure-name)
      ```

      to suspend queue manager access to the SMDS. During this time the data on the shared queues associated with the SMDS will be temporarily unavailable.

   b. Following the process in "Removing data set encryption from a data set" on page 463 for each data set which makes up the SMDS.

   c. Issuing the command

      ```
      RESET SMDS(*) ACCESS(ENABLED) CFSTRUCT(structure-name)
      ```

      to resume queue manager access to the SMDS.

Using z/OS data set encryption with a queue manager that does not support it

If you accidentally backwards migrate a queue manager to a version of IBM MQ for z/OS that does not support data set encryption, and forget to remove the encryption policies and decrypt the data you get an error when the queue manager tries to access the data set.

The error depends on the data set type and is shown in the following table.

Note: If one or more of these errors occur, you need to follow the processes described in "Removing data set encryption from a data set" on page 463 for the affected data set. These can be performed without changing the version of IBM MQ for z/OS.

Data set	Error if queue manager does not support z/OS data set encryption
Page set 0	Abend 5C6-00C91400 at queue manager start
Page sets 1-99	MQRC 2193 "Pageset error" when accessing page set, for example, on MQPUT
Active log	Abend 5C6-00E80084 at queue manager start
SMDS	Message IEC161I-122 logged "The data set has a KEYLABEL, but the user did not specify that the application could handle encryption". SMDS marked AVAIL(ERROR).

Data integrity of messages

To maintain data integrity, you can use various types of user exit program to provide message digests or digital signatures for your messages.

Data integrity

Implementing data integrity in messages

When you use TLS, your choice of CipherSpec determines the level of data integrity in the enterprise. If you use the IBM MQ Advanced Message Service (AMS) you can specify the integrity for a unique message.

Implementing data integrity in message exits

A message can be digitally signed by a message exit at the sending end of a channel. The digital signature can then be checked by a message exit at the receiving end of a channel to detect whether the message has been deliberately modified.

Some protection can be provided by using a message digest instead of a digital signature. A message digest might be effective against casual or indiscriminate tampering, but it does not prevent the more informed individual from changing or replacing the message, and generating a completely new digest for it. This is particularly true if the algorithm that is used to generate the message digest is a well known one.

Implementing data integrity in send and receive exits

On a message channel, message exits are more appropriate for providing this service because a message exit has access to a whole message. On an MQI channel, parameters on MQI calls might contain application data that needs to be protected and only send and receive exits can provide this protection.

Implementing data integrity in the API exit or API-crossing exit

A message can be digitally signed by an API or API-crossing exit when the message is put by the sending application. The digital signature can then be checked by a second exit when the message is retrieved by the receiving application to detect whether the message has been deliberately modified.

Some protection can be provided by using a message digest instead of a digital signature. A message digest might be effective against casual or indiscriminate tampering, but it does not prevent the more informed individual from changing or replacing the message, and generating a completely new digest for it. This is particularly true if the algorithm that is used to generate the message digest is a well known one,

Further information

See the section on "Enabling CipherSpecs" on page 411 for more information on ensuring data integrity.

Related tasks

Connecting two queue managers using TLS

Connecting a client to a queue manager securely

Auditing

You can check for security intrusions, or attempted intrusions, by using event messages. You can also check the security of your system by using the IBM MQ Explorer.

To detect attempts to perform unauthorized actions such as connecting to a queue manager or put a message on a queue, inspect the event messages produced by your queue managers, particularly authority event messages. For more information about queue manager event messages, see Queue manager events, and for more information about event monitoring in general, see Event monitoring.

Keeping clusters secure

Authorize or prevent queue managers joining clusters or putting messages on cluster queues. Force a queue manager to leave a cluster. Take account of some additional considerations when configuring TLS for clusters.

Stopping unauthorized queue managers sending messages

Prevent unauthorized queue managers sending messages to your queue manager using a channel security exit.

Before you begin

Clustering has no effect on the way security exits work. You can restrict access to a queue manager in the same way as you would in a distributed queuing environment.

About this task

Prevent selected queue managers from sending messages to your queue manager:

Procedure

1. Define a channel security exit program on the CLUSRCVR channel definition.
2. Write a program that authenticates queue managers trying to send messages on your cluster-receiver channel and denies them access if they are not authorized.

What to do next

Channel security exit programs are called at MCA initiation and termination.

Stopping unauthorized queue managers putting messages on your queues

Use the channel put authority attribute on the cluster-receiver channel to stop unauthorized queue managers putting messages on your queues. Authorize a remote queue manager by checking the user ID in the message using RACF on z/OS, or the OAM on Multiplatforms.

About this task

Use the security facilities of a platform and the access control mechanism in IBM MQ to control access to queues.

Procedure

1. To prevent certain queue managers from putting messages on a queue, use the security facilities available on your platform.

   For example:

   - z/OS RACF or other external security managers on IBM MQ for z/OS

- Multi The object authority manager (OAM) on other Multiplatforms.

2. Use the put authority, PUTAUT, attribute on the CLUSRCVR channel definition.

   The PUTAUT attribute allows you to specify what user identifiers are to be used to establish authority to put a message to a queue.

   The options on the PUTAUT attribute are:

   DEF
   Use the default user ID.

   z/OS On z/OS, the check might involve using both the user ID received from the network and that derived from MCAUSER.

   CTX
   Use the user ID in the context information associated with the message.

   z/OS On z/OS the check might involve using either the user ID received from the network, or that derived from MCAUSER, or both. Use this option if the link is trusted and authenticated.

   z/OS **ONLYMCA (z/OS only)**
   As for DEF, but any user ID received from the network is not used. Use this option if the link is not trusted. You want to allow only a specific set of actions on it, which are defined for the MCAUSER.

   z/OS **ALTMCA (z/OS only)**
   As for CTX, but any user ID received from the network is not used.

Authorizing putting messages on remote cluster queues

On z/OS set up authorization to put to a cluster queue using RACF. On Multiplatforms, authorize access to connect to the queue managers, and to put to the queues on those queue managers.

About this task

The default behavior is to perform access control against the SYSTEM.CLUSTER.TRANSMIT.QUEUE. Note that this behavior applies, even if you are using multiple transmission queues.

The specific behavior described in this topic applies only when you have configured the **ClusterQueueAccessControl** attribute in the qm.ini file to be *RQMName*, as described in the Security stanza topic, and restarted the queue manager.

Procedure

- z/OS
  For z/OS, issue the following commands:

```
RDEFINE MQQUEUE QMgrName.QUEUE. QueueName UACC(NONE)
PERMIT QMgrName.QUEUE. QueueName CLASS(MQADMIN) ID(GroupName) ACCESS(UPDATE)
```

- ALW
  For AIX, Linux, and Windows systems, issue the following commands:

```
setmqaut -m QMgrName -t qmgr -g GroupName +connect
setmqaut -m QMgrName -t queue -n QueueName -g GroupName -all +put
```

- IBM i
  For IBM i, issue the following commands:

```
GRTMQMAUT OBJ(' QMgrName ') OBJTYPE(*MQM) USER(GroupName) AUT(*CONNECT)
GRTMQMAUT OBJ(' QueueName ') OBJTYPE(*Q) USER(GroupName) AUT(*PUT) MQMNAME(' QMgrName ')
```

The user can put messages only to the specified cluster queue, and no other cluster queues.

The variable names have the following meanings:

QMgrName
The name of the queue manager. On z/OS, this value can also be the name of a queue sharing group.

GroupName
The name of the group to be granted access.

QueueName
Name of the queue or generic profile for which to change authorizations.

What to do next

If you specify a reply-to queue when you put a message on a cluster queue, the consuming application must have authority to send the reply. Set this authority by following the instructions in "Granting authority to put messages to a remote cluster queue" on page 387.

Related concepts

Security stanza in qm.ini

Preventing queue managers joining a cluster

If a rogue queue manager joins a cluster it is difficult to prevent it receiving messages you do not want it to receive.

Procedure

If you want to ensure that only certain authorized queue managers join a cluster you have a choice of three techniques:

- Using channel authentication records you can block the cluster channel connection based on: the remote IP address, the remote queue manager name, or the TLS Distinguished Name provided by the remote system.
- Write an exit program to prevent unauthorized queue managers from writing to `SYSTEM.CLUSTER.COMMAND.QUEUE`. Do not restrict access to `SYSTEM.CLUSTER.COMMAND.QUEUE` such that no queue manager can write to it, or you would prevent any queue manager from joining the cluster.
- A security exit program on the `CLUSRCVR` channel definition.

Security exits on cluster channels

Extra considerations when using security exits on cluster channels.

About this task

When a cluster-sender channel is first started, it uses attributes defined manually by a system administrator. When the channel is stopped and restarted, it picks up the attributes from the corresponding cluster-receiver channel definition. The original cluster-sender channel definition is overwritten with the new attributes, including the `SecurityExit` attribute.

Procedure

1. You must define a security exit on both the cluster-sender end and the cluster-receiver end of a channel.

The initial connection must be made with a security-exit handshake, even though the security exit name is sent over from the cluster-receiver definition.

2. Validate the `PartnerName` in the `MQCXP` structure in the security exit.

   The exit must allow the channel to start only if the partner queue manager is authorized
3. Design the security exit on the cluster-receiver definition to be receiver initiated.
4. If you design it as sender initiated, an unauthorized queue manager without a security exit can join the cluster because no security checks are performed.

   Not until the channel is stopped and restarted can the `SCYEXIT` name be sent over from the cluster-receiver definition and full security checks made.
5. To view the cluster-sender channel definition that is currently in use, use the command:

   ```
   DISPLAY CLUSQMGR( queue manager ) ALL
   ```

   The command displays the attributes that have been sent across from the cluster-receiver definition.
6. To view the original definition, use the command:

   ```
   DISPLAY CHANNEL( channel name ) ALL
   ```
7. You might need to define a channel auto-definition exit, `CHADEXIT`, on the cluster-sender queue manager, if the queue managers are on different platforms.

   Use the channel auto-definition exit to set the `SecurityExit` attribute to an appropriate format for the target platform.
8. Deploy and configure the security-exit.

   z/OS **z/OS**

   The security-exit load module must be in the data set specified in the `CSQXLIB DD` statement of the channel-initiator address-space procedure.

   ALW **AIX, Linux, and Windows systems**

   - The security-exit dynamic link library must be in the path specified in the `SCYEXIT` attribute of the channel definition.
   - The channel auto-definition exit dynamic link library must be in the path specified in the `CHADEXIT` attribute of the queue manager definition.

Forcing unwanted queue managers to leave a cluster

Force an unwanted queue manager to leave a cluster by issuing the `RESET CLUSTER` command at a full repository queue manager.

About this task

You can force an unwanted queue manager to leave a cluster. If for example, a queue manager is deleted but its cluster-receiver channels are still defined to the cluster. You might want to tidy up.

Only full repository queue managers are authorized to eject a queue manager from a cluster.

Note: Although using the RESET CLUSTER command forcibly removes a queue manager from a cluster, the use of RESET CLUSTER by itself does not prevent the queue manager rejoining the cluster later. To ensure that the queue manager does not rejoin the cluster, follow the steps detailed in "Preventing queue managers joining a cluster" on page 468.

Follow this procedure to eject the queue manager `OSLO` from the cluster `NORWAY`:

Procedure

1. On a full repository queue manager, issue the command:

```
RESET CLUSTER(NORWAY) QMNAME(OSLO) ACTION(FORCEREMOVE)
```

2. Alternative use the `QMID` instead of `QMNAME` in the command:

```
RESET CLUSTER(NORWAY) QMID(qmid) ACTION(FORCEREMOVE)
```

Note: `QMID` is a string, so the value of `qmid` should be surrounded by single quotation marks, for example, `QMID('FR01_2019-07-15_14.42.42')`.

Results

The queue manager that is force removed does not change; its local cluster definitions show it to be in the cluster. The definitions at all other queue managers do not show it in the cluster.

Preventing queue managers receiving messages

You can prevent a cluster queue manager from receiving messages it is unauthorized to receive by using exit programs.

About this task

It is difficult to stop a queue manager that is a member of a cluster from defining a queue. There is a danger that a rogue queue manager joins a cluster, and defines its own instance of one of the queues in the cluster. It can now receive messages that it is not authorized to receive. To prevent a queue manager receiving messages, use one of the following options given in the procedure.

Procedure

- A channel exit program on each cluster-sender channel. The exit program uses the connection name to determine the suitability of the destination queue manager to be sent the messages.
- A cluster workload exit program, which uses the destination records to determine the suitability of the destination queue and queue manager to be sent the messages.

SSL/TLS and clusters

When configuring TLS for clusters, be aware a CLUSRCVR channel definition is propagated to other queue managers as an auto-defined CLUSSDR channel. If a CLUSRCVR channel uses TLS, you must configure TLS on all queue managers that communicate using the channel.

For more information about TLS, see "TLS security protocols in IBM MQ " on page 24. The advice there is generally applicable to cluster channels, but you might want to give some special consideration to the following:

In an IBM MQ cluster a particular `CLUSRCVR` channel definition is frequently propagated to many other queue managers where it is transformed into an auto-defined `CLUSSDR`. Subsequently the auto-defined `CLUSSDR` is used to start a channel to the `CLUSRCVR`. If the `CLUSRCVR` is configured for TLS connectivity the following considerations apply:

- All queue managers that want to communicate with this `CLUSRCVR` must have access to TLS support. This TLS provision must support the CipherSpec for the channel.
- The different queue managers to which the auto-defined cluster-sender channels have been propagated will each have a different distinguished name associated. If distinguished name peer checking is to be used on the `CLUSRCVR` it must be set up so all of the distinguished names that can be received are successfully matched.

For example, let us assume that all of the queue managers that will host cluster-sender channels which will connect to a particular CLUSRCVR, have certificates associated. Let us also assume that the distinguished names in all of these certificates define the country as UK, organization as IBM, the organization unit as IBM MQ Development, and all have common names in the form DEVT.QMnnn, where nnn is numeric.

In this case an SSLPEER value of `C=UK, O=IBM, OU=IBM MQ Development, CN=DEVT.QM*` on the CLUSRCVR will allow all the required cluster-sender channels to connect successfully, but will prevent unwanted cluster-sender channels from connecting.

- If custom CipherSpec strings are used, be aware that the custom string formats are not allowed on all platforms. An example of this is that the `CipherSpec` string RC4_SHA_US has a value of 05 on IBM i but is not a valid specification on AIX, Linux, and Windows systems. So if custom SSLCIPH parameters are used on a CLUSRCVR, all resulting auto-defined cluster-sender channels should reside on platforms on which the underlying TLS support implements this `CipherSpec` and on which it can be specified with the custom value. If you cannot select a value for the SSLCIPH parameter that will be understood throughout your cluster you will need a channel auto definition exit to change it into something the platforms being used will understand. Use the textual `CipherSpec` strings where possible (for example TLS_RSA_WITH_AES_128_CBC_SHA).

An SSLCRLNL parameter applies to an individual queue manager and is not propagated to other queue managers within a cluster.

Upgrading clustered queue managers and channels to SSL/TLS

Upgrade the cluster channels one at a time, changing all the CLUSRCVR channels before the CLUSSDR channels.

Before you begin

Consider the following considerations, as these might affect your choice of CipherSpec for a cluster:

- Some CipherSpecs are not available on all platforms. Take care to choose a CipherSpec that is supported by all of the queue managers in the cluster.
- Some CipherSpecs might be new in the current IBM MQ release and not supported in older releases. A cluster containing queue managers running at different MQ releases is only be able to use the CipherSpecs supported by each release.

  To use a new CipherSpec within a cluster, you must first migrate all of the cluster queue managers to the current release.
- Some CipherSpecs require a specific type of digital certificate to be used, notably those that use Elliptic Curve Cryptography.

Attention: It is not possible to use a mixture of Elliptic Curve-signed certificates and RSA-signed certificates on queue managers that you want to join together as part of a cluster.

Queue managers in a cluster must all use RSA-signed certificates, or all use EC-signed certificates, not a mixture of both.

See "Digital certificates and CipherSpec compatibility in IBM MQ" on page 47 for more information.

Upgrade all queue managers in the cluster to IBM MQ V8 or higher, if they are not already at these levels. Distribute the certificates and keys so that TLS works from each of them.

Before you can upgrade to, or use, any of the alias CipherSpecs (ANY_TLS13, ANY_TLS13_OR_HIGHER, ANY_TLS12, ANY_TLS12_OR_HIGHER, and so on) you must upgrade your queue managers:

- Multi Upgrade all IBM MQ for Multiplatforms queue managers in the cluster to IBM MQ 9.1.4 or later.
- z/OS Upgrade all IBM MQ for z/OS queue managers in the cluster to IBM MQ for z/OS 9.2.0 or later.

you must

About this task

Change the CLUSRCVR channels before the CLUSSDR channels.

Procedure

1. Switch the CLUSRCVR channels to TLS in any order you like, changing one CLUSRCVR at a time, and allow the changes to flow through the cluster before changing the next.

   Important: Make sure that you do not change the reverse path until the changes for the current channel have been distributed throughout the cluster.
2. Optional: Switch all manual CLUSSDR channels to TLS.

   This does not have any effect on the operation of the cluster, unless you use the REFRESH CLUSTER command with the REPOS(YES) option.

   Note: For large clusters, use of the **REFRESH CLUSTER** command can be disruptive to the cluster while it is in progress, and again at 27 day intervals thereafter when the cluster objects automatically send status updates to all interested queue managers. See Refreshing in a large cluster can affect performance and availability of the cluster.
3. Use the DISPLAY CLUSQMGR command to ensure that the new security configuration has been propagated throughout the cluster.
4. Restart the channels to use TLS and run REFRESH SECURITY (SSL).

Related concepts

"Enabling CipherSpecs" on page 411
Enable a CipherSpec by using the **SSLCIPH** parameter in either the **DEFINE CHANNEL** or **ALTER CHANNEL** MQSC command.

"Digital certificates and CipherSpec compatibility in IBM MQ" on page 47
This topic provides information on how to choose appropriate CipherSpecs and digital certificates for your security policy, by outlining the relationship between CipherSpecs and digital certificates in IBM MQ.

Related information

Clustering: Using REFRESH CLUSTER best practices

Disabling SSL/TLS on clustered queue managers and channels

To turn off TLS, set the SSLCIPH parameter to ' '. Disable TLS on the cluster channels individually, changing all the cluster receiver channels before the cluster sender channels.

About this task

Change one cluster receiver channel at a time, and allow the changes to flow through the cluster before changing the next.

Important: Ensure that you do not change the reverse path until the changes for the current channel have been distributed throughout the cluster.

Procedure

1. Set the value of the SSLCIPH parameter to ' ', an empty string in a single quotation mark IBM i, or *NONE on IBM i .

   You can turn off TLS on the cluster receiver channels in any order you like.

   Note that the changes flow in the opposite direction over channels on which you leave TLS active.
2. Check that the new value is reflected in all the other queue managers by using the command **DISPLAY CLUSQMGR(*)** ALL.
3. Turn off TLS on all manual cluster sender channels.

This does not have any effect on the operation of the cluster, unless you use the **REFRESH CLUSTER** command with the REPOS(YES) option.

For large clusters, use of the **REFRESH CLUSTER** command can be disruptive to the cluster while it is in progress, and again at regular intervals thereafter, when the cluster objects automatically send status updates to all interested queue managers. See Refreshing in a large cluster can affect performance and availability of the cluster for more information.

4. Stop and restart the cluster sender channels.

Publish/subscribe security

The components and interactions that are involved in publish/subscribe are described as an introduction to the more detailed explanations and examples that follow.

There are a number of components involved in publishing and subscribing to a topic. Some of the security relationships between them are illustrated in Figure 22 on page 473 and described in the following example.

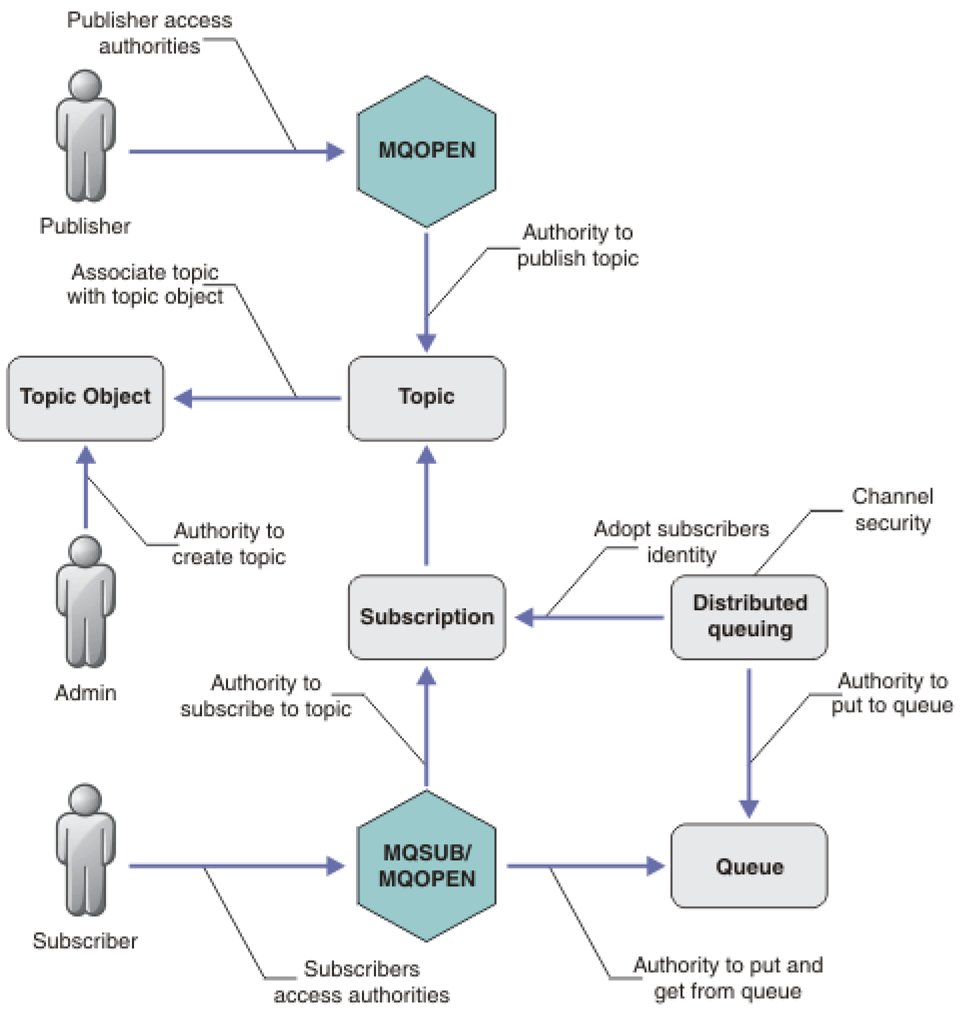

Publisher access authorities
Publisher
MQOPEN
Authority to publish topic
Associate topic with topic object
Topic Object
Topic
Authority to create topic
Admin
Channel security
Adopt subscribers identity
Subscription
Distributed queuing
Authority to subscribe to topic
Authority to put to queue
Subscriber
MQSUB/ MQOPEN
Queue
Subscribers access authorities
Authority to put and get from queue

Figure 22. Publish/subscribe security relationships between various components

Topics

Topics are identified by topic strings, and are typically organized into trees, see Topic trees. You need to associate a topic with a topic object to control access to the topic. "Topic security model" on page 475 explains how you secure topics using topic objects.

Administrative topic objects

You can control who has access to a topic, and for what purpose, by using the command **`setmqaut`** with a list of administrative topic objects. See the examples, "Grant access to a user to subscribe to a topic" on page 480 and "Grant access to a user to publish to a topic" on page 487.

z/OS For controlling access to topic objects on z/OS, see Profiles for topic security.

Subscriptions

Subscribe to one or more topics by creating a subscription supplying a topic string, which can include wildcards, to match against the topic strings of publications. For further details, see:

Subscribe using a topic object
"Subscribing using the topic object name" on page 476

Subscribe using a topic
"Subscribing using a topic string where the topic node does not exist" on page 477

Subscribe using a topic with wildcards
"Subscribing using a topic string that contains wildcard characters" on page 478

A subscription contains information about the identity of the subscriber and the identity of the destination queue on to which the publications are to be placed. It also contains information about how the publication is to be placed on the destination queue.

As well as defining which subscribers have the authority to subscribe to certain topics, you can restrict subscriptions to being used by an individual subscriber. You can also control what information about the subscriber is used by the queue manager when publications are placed on to the destination queue. See "Subscription security" on page 492.

Queues

The destination queue is an important queue to secure. It is local to the subscriber, and publications that matched the subscription are placed onto it. You need to consider access to the destination queue from two perspectives:

1. Putting a publication on to the destination queue.
2. Getting the publication off the destination queue.

The queue manager puts a publication onto the destination queue using an identity provided by the subscriber. The subscriber, or a program that has been delegated the task of getting publications, takes messages off the queue. See "Authority to destination queues" on page 478.

There are no topic object aliases, but you can use an alias queue as the alias for a topic object. If you do so, as well as checking authority to use the topic for publish or subscribe, the queue manager checks authority to use the queue.

"Publish/subscribe security between queue managers" on page 494

Your permission to publish or subscribe to a topic is checked on the local queue manager using local identities and authorizations. Authorization does not depend on whether the topic is defined or not, nor where it is defined. Consequently, you need to perform topic authorization on every queue manager in a cluster when clustered topics are used.

Note: The security model for topics differs from the security model for queues. You can achieve the same result for queues by defining a queue alias locally for every clustered queue.

Queue managers exchange subscriptions in a cluster. In most IBM MQ cluster configurations, channels are configured with PUTAUT=DEF to place messages onto target queues using the authority of the channel process. You can modify the channel configuration to use PUTAUT=CTX to require the subscribing user to have authority to propagate a subscription onto another queue manager in a cluster.

"Publish/subscribe security between queue managers" on page 494 describes how to change your channel definitions to control who is allowed to propagate subscriptions onto other servers in the cluster.

Authorization

You can apply authorization to topic objects, just like queues and other objects. There are three authorization operations, pub, sub, and `resume` that you can apply only to topics. The details are described in Specifying authorities for different object types.

Function calls

In publish and subscribe programs, like in queued programs, authorization checks are made when objects are opened, created, changed, or deleted. Checks are not made when MQPUT or MQGET MQI calls are made to put and get publications.

To publish a topic, perform an MQOPEN on the topic, which performs the authorization checks. Publish messages to the topic handle using the MQPUT command, which performs no authorization checks.

To subscribe to a topic, typically you perform an MQSUB command to create or resume the subscription, and also to open the destination queue to receive publications. Alternatively, perform a separate MQOPEN to open the destination queue, and then perform the MQSUB to create or resume the subscription.

Whichever calls you use, the queue manager checks that you can subscribe to the topic and get the resulting publications from the destination queue. If the destination queue is unmanaged, authorization checks are also made that the queue manager is able to place publications on the destination queue. It uses the identity it adopted from a matching subscription. It is assumed that the queue manager is always able to place publications onto managed destination queues.

Roles

Users are involved in four roles in running publish/subscribe applications:

1. Publisher
2. Subscriber
3. Topic administrator
4. IBM MQ Administrator - member of group mqm

Define groups with appropriate authorizations corresponding to the publish, subscribe, and topic administration roles. You can then assign principals to these groups authorizing them to perform specific publish and subscribe tasks.

In addition, you need to extend the administrative operations authorizations to the administrator of the queues and channels responsible for moving publications and subscriptions.

Topic security model

Only defined topic objects can have associated security attributes. For a description of topic objects, see Administrative topic objects. The security attributes specify whether a specified user ID, or security group, is permitted to perform a subscribe or a publish operation on each topic object.

The security attributes are associated with the appropriate administration node in the topic tree. When an authority check is made for a particular user ID during a subscribe or publish operation, the authority granted is based on the security attributes of the associated topic tree node.

The security attributes are an access control list, indicating what authority a particular operating system user ID or security group has to the topic object.

Consider the following example where the topic objects have been defined with the security attributes, or authorities shown:

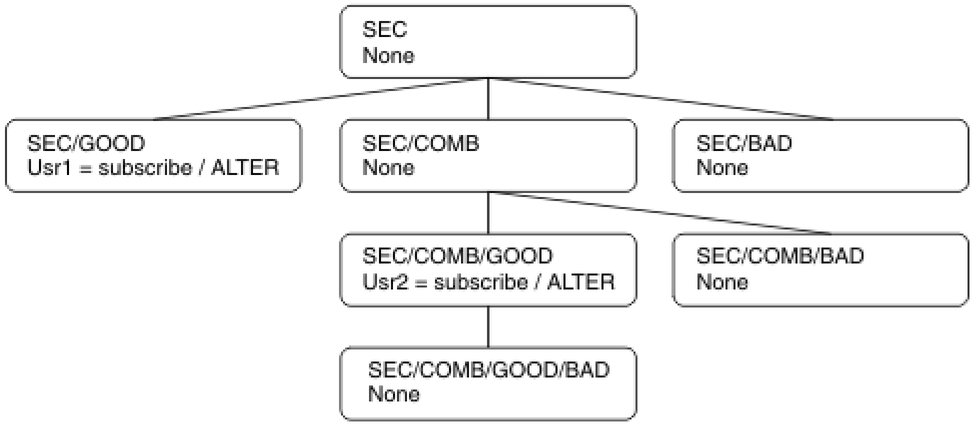
Table 87. Example topic object authorities

Topic name	Topic string	Authorities - Multiplatforms	z/OS authorities
SECROOT	SEC	None	None

Table 87. Example topic object authorities (continued)

Topic name	Topic string	Authorities - Multiplatforms	z/OS authorities
SECGOOD	SEC/GOOD	usr1+subscribe	ALTER HLQ.SUBSCRIBE.SECGOOD
SECBAD	SEC/BAD	None	None HLQ.SUBSCRIBE.SECBAD
SECCOMB	SEC/COMB	None	None HLQ.SUBSCRIBE.SECCOMB
SECCOMBB	SEC/COMB/GOOD/BAD	None	None HLQ.SUBSCRIBE.SECCOMBB
SECCOMBG	SEC/COMB/GOOD	usr2+subscribe	ALTER HLQ.SUBSCRIBE.SECCOMBG
SECCOMBN	SEC/COMB/BAD	None	None HLQ.SUBSCRIBE.SECCOMBN

The topic tree with the associated security attributes at each node can be represented as follows:



The examples listed give the following authorizations:

- At the root node of the tree /SEC, no user has authority at that node.
- usr1 has been granted subscribe authority to the object /SEC/GOOD
- usr2 has been granted subscribe authority to the object /SEC/COMB/GOOD

Subscribing using the topic object name

When subscribing to a topic object by specifying the MQCHAR48 name, the corresponding node in the topic tree is located. If the security attributes associated with the node indicate that the user has authority to subscribe, then access is granted.

If the user is not granted access, the parent node in the tree determines if the user has authority to subscribe at the parent node level. If so, then access is granted. If not, then the parent of that node is considered. The recursion continues until a node is located that grants subscribe authority to the user.

The recursion stops when the root node is considered without authority having been granted. In the latter case, access is denied.

In short, if any node in the path grants authority to subscribe to that user or application, the subscriber is allowed to subscribe at that node, or anywhere below that node in the topic tree.

The root node in the example is SEC.

The user is granted subscribe authority if the access control list indicates that the user ID itself has authority, or that an operating system security group of which the user ID is a member has authority.

So, for example:

- If `usr1` tries to subscribe, using a topic string of `SEC/GOOD`, the subscription would be allowed as the user ID has access to the node associated with that topic. However, if `usr1` tried to subscribe using topic string `SEC/COMB/GOOD` the subscription would not be allowed as the user ID does not have access to the node associated with it.
- If `usr2` tries to subscribe, using a topic string of `SEC/COMB/GOOD` the subscription would be allowed to as the user ID has access to the node associated with the topic. However, if `usr2` tried to subscribe to `SEC/GOOD` the subscription would not be allowed as the user ID does not have access to the node associated with it.
- If `usr2` tries to subscribe using a topic string of `SEC/COMB/GOOD/BAD` the subscription would be allowed to because the user ID has access to the parent node `SEC/COMB/GOOD`.
- If `usr1` or `usr2` tries to subscribe using a topic string of `/SEC/COMB/BAD`, neither would be allowed as they do not have access to the topic node associated with it, or the parent nodes of that topic.

A subscribe operation specifying the name of a topic object that does not exist results in an MQRC_UNKNOWN_OBJECT_NAME error.

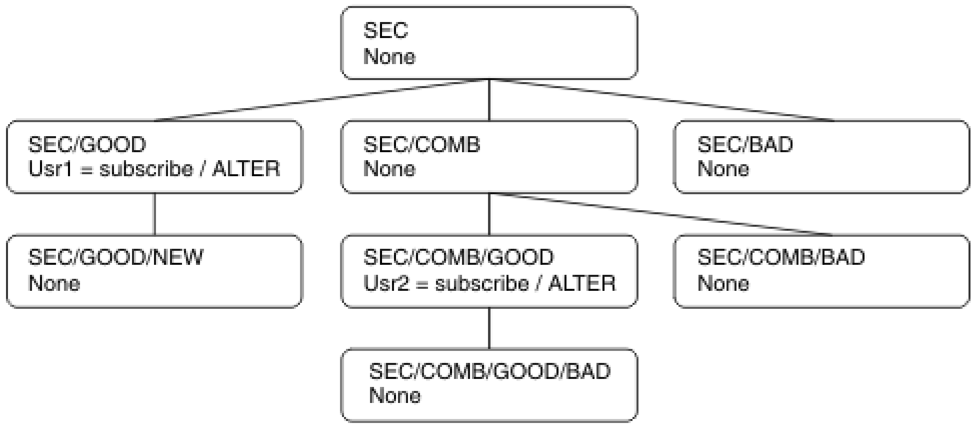
Subscribing using a topic string where the topic node exists

The behavior is the same as when specifying the topic by the MQCHAR48 object name.

Subscribing using a topic string where the topic node does not exist

Consider the case of an application subscribing, specifying a topic string representing a topic node that does not currently exist in the topic tree. The authority check is performed as outlined in the previous section. The check starts with the parent node of that which is represented by the topic string. If the authority is granted, a new node representing the topic string is created in the topic tree.

For example, `usr1` tries to subscribe to a topic `SEC/GOOD/NEW`. Authority is granted as `usr1` has access to the parent node `SEC/GOOD`. A new topic node is created in the tree as the following diagram shows. The new topic node is not a topic object it does not have any security attributes associated with it directly; the attributes are inherited from its parent.



Subscribing using a topic string that contains wildcard characters

Consider the case of subscribing using a topic string that contains a wildcard character. The authority check is made against the node in the topic tree that matches the fully qualified part of the topic string.

So, if an application subscribes to SEC/COMB/GOOD/*, an authority check is carried out as outlined in the previous two sections on the node SEC/COMB/GOOD in the topic tree.

Similarly, if an application needs to subscribe to SEC/COMB/*/GOOD, an authority check is carried out on the node SEC/COMB.

Authority to destination queues

When subscribing to a topic, one of the parameters is the handle hobj of a queue that has been opened for output to receive the publications.

If hobj is not specified, but is blank, a managed queue is created if the following conditions apply:

- The MQSO_MANAGED option has been specified.
- The subscription does not exist.
- Create is specified.

If hobj is blank, and you are altering or resuming an existing subscription, the previously provided destination queue could be either managed or unmanaged.

The application or user making the MQSUB request must have the authority to put messages to the destination queue it has provided; in effect authority to have published messages put on that queue. The authority check follows the existing rules for queue security checking.

The security checking includes alternate user ID and context security checks where required. To be able to set any of the Identity context fields you must specify the MQSO_SET_IDENTITY_CONTEXT option as well as the MQSO_CREATE or MQSO_ALTER option. You cannot set any of the Identity context fields on an MQSO_RESUME request.

If the destination is a managed queue, no security checks are performed against the managed destination. If you are allowed to subscribe to a topic it is assumed that you can use managed destinations.

Publishing using the topic name or topic string where the topic node exists

The security model for publishing is the same as that for subscribing, with the exception of wildcards. Publications do not contain wildcards; so there is no case of a topic string containing wildcards to consider.

The authorities to publish and subscribe are distinct. A user or group can have the authority to do one without necessarily being able to do the other.

When publishing to a topic object by specifying either the MQCHAR48 name or the topic string, the corresponding node in the topic tree is located. If the security attributes associated with the topic node indicates that the user has authority to publish, then access is granted.

If access is not granted, the parent node in the tree determines if the user has authority to publish at that level. If so, then access is granted. If not, the recursion continues until a node is located which grants publish authority to the user. The recursion stops when the root node is considered without authority having been granted. In the latter case, access is denied.

In short, if any node in the path grants authority to publish to that user or application, the publisher is allowed to publish at that node or anywhere below that node in the topic tree.

Publishing using the topic name or topic string where the topic node does not exist

As with the subscribe operation, when an application publishes, specifying a topic string representing a topic node that does not currently exist in the topic tree, the authority check is performed starting with the parent of the node represented by the topic string. If the authority is granted, a new node representing the topic string is created in the topic tree.

Publishing using an alias queue that resolves to a topic object

If you publish using an alias queue that resolves to a topic object then security checking occurs on both the alias queue and the underlying topic to which it resolves.

The security check on the alias queue verifies that the user has authority to put messages on that alias queue and the security check on the topic verifies that the user can publish to that topic. When an alias queue resolves to another queue, checks are not made on the underlying queue. Authority checking is performed differently for topics and queues.

Closing a subscription

There is additional security checking if you close a subscription using the `MQCO_REMOVE_SUB` option if you did not create the subscription under this handle.

A security check is performed to ensure that you have the correct authority to do this as the action results in the removal of the subscription. If the security attributes associated with the topic node indicate that the user has authority, then access is granted. If not, then the parent node in the tree is considered to determine if the user has authority to close the subscription. The recursion continues until either authority is granted or the root node is reached.

Defining, altering, and deleting a subscription

No subscribe security checks are performed when a subscription is created administratively, rather than using an `MQSUB` API request. The administrator has already been given this authority through the command.

Security checks are performed to ensure that publications can be put on the destination queue associated with the subscription. The checks are performed in the same way as for an `MQSUB` request.

The user ID that is used for these security checks depends upon the command being issued. If the **SUBUSER** parameter is specified it affects the way the check is performed, as shown in Table 88 on page 479:

Table 88. User IDs used for security checks for commands

Command	SUBUSER specified and blank	SUBUSER specified and completed	SUBUSER not specified
	Use the administrator ID		Use the user ID from the LIKE subscription
	Use the administrator ID		Use the.DEFAULT.SU user IDB from thesubscription SYSTEM- if blank, use the administrator ID
	Use the administrator ID		Use the user ID from the existing subscription

The only security check performed when deleting subscriptions using the DELETE SUB command is the command security check.

Example publish/subscribe security setup

This section describes a scenario that has access control set up on topics in a way that allows the security control to be applied as required.

Grant access to a user to subscribe to a topic

This topic is the first one in a list of tasks that tells you how to grant access to topics by more than one user.

About this task

This task assumes that no administrative topic objects exist, nor have any profiles been defined for subscription or publication. The applications are creating new subscriptions, rather than resuming existing ones, and are doing so using the topic string only.

An application can make a subscription by providing a topic object, or a topic string, or a combination of both. Whichever way the application selects, the effect is to make a subscription at a certain point in the topic tree. If this point in the topic tree is represented by an administrative topic object, a security profile is checked based on the name of that topic object.

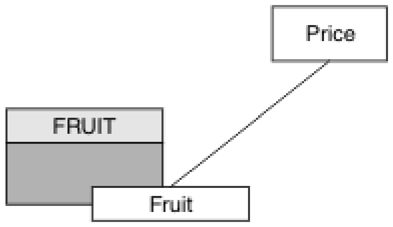


Figure 23. Topic object access example

Table 89. Example topic object access

Topic	Subscribe access required	Topic object
Price	No user	None
Price/Fruit	USER1	FRUIT

Define a new topic object as follows:

Procedure

1. Issue the MQSC command `DEF TOPIC(FRUIT) TOPICSTR('Price/Fruit')`.
2. Grant access as follows:
   - z/OS **z/OS** :

     Grant access to USER1 to subscribe to topic "`Price/Fruit`" by granting the user access to the `hlq.SUBSCRIBE.FRUIT` profile. Do this, using the following RACF commands:

     ```
     RDEFINE MXTOPIC hlq.SUBSCRIBE.FRUIT UACC(NONE)
     PERMIT hlq.SUBSCRIBE.FRUIT CLASS(MXTOPIC) ID(USER1) ACCESS(ALTER)
     ```

   - Multi **Multiplatforms**:

Grant access to USER1 to subscribe to topic "`Price/Fruit`" by granting the user access to the FRUIT object. Do this, using the authorization command for the platform:

ALW **AIX, Linux, and Windows systems**

```
setmqaut -t topic -n FRUIT -p USER1 +sub
```

IBM i **IBM i**

```
GRTMQAUT OBJ(FRUIT) OBJTYPE(*TOPIC) USER(USER1) AUT(*SUB)
```

Results

When USER1 attempts to subscribe to topic "`Price/Fruit`" the result is success.

When USER2 attempts to subscribe to topic "`Price/Fruit`" the result is failure with an `MQRC_NOT_AUTHORIZED` message, together with:

- z/OS On z/OS, the following messages seen on the console that show the full security path through the topic tree that has been attempted:

```
ICH408I USER(USER2   ) ...
  hlq.SUBSCRIBE.FRUIT ...

ICH408I USER(USER2   ) ...
  hlq.SUBSCRIBE.SYSTEM.BASE.TOPIC ...
```

- ALW On AIX, Linux, and Windows, the following authorization event:

```
MQRC_NOT_AUTHORIZED
ReasonQualifier      MQRQ_SUB_NOT_AUTHORIZED
UserIdentifier        USER2
AdminTopicNames      FRUIT, SYSTEM.BASE.TOPIC
TopicString           "Price/Fruit"
```

- IBM i On IBMi, the following authorization event:

```
MQRC_NOT_AUTHORIZED
ReasonQualifier      MQRQ_SUB_NOT_AUTHORIZED
UserIdentifier        USER2
AdminTopicNames      FRUIT, SYSTEM.BASE.TOPIC
TopicString           "Price/Fruit"
```

Note that this is an illustration of what you see; not all the fields.

Grant access to a user to subscribe to a topic deeper within the tree

This topic is the second in a list of tasks that tells you how to grant access to topics by more than one user.

Before you begin

This topic uses the setup described in "Grant access to a user to subscribe to a topic" on page 480.

About this task

If the point in the topic tree where the application makes the subscription is not represented by an administrative topic object, move up the tree until the closest parent administrative topic object is located. The security profile is checked, based on the name of that topic object.

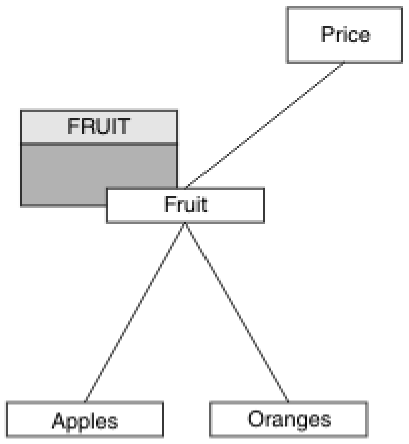


Figure 24. Example of granting access to a topic within a topic tree

Table 90. Access requirements for example topics and topic objects

Topic	Subscribe access required	Topic object
Price	No user	None
Price/Fruit	USER1	FRUIT
Price/Fruit/ Apples	USER1	
Price/Fruit/ Oranges	USER1	

In "Grant access to a user to subscribe to a topic" on page 480, USER1 was granted access to subscribe to topic "`Price/Fruit`" by granting it access to the `hlq.SUBSCRIBE.FRUIT` profile on z/OS and subscribe access to the FRUIT profile on Multiplatforms. This single profile also grants USER1 access to subscribe to "`Price/Fruit/Apples`", "`Price/Fruit/Oranges`" and "`Price/Fruit/#`".

When USER1 attempts to subscribe to topic "`Price/Fruit/Apples`" the result is success.

When USER2 attempts to subscribe to topic "`Price/Fruit/Apples`" the result is failure with an `MQRC_NOT_AUTHORIZED` message, together with:

- z/OS On z/OS, the following messages seen on the console that show the full security path through the topic tree that has been attempted:

```
ICH408I USER(USER2   ) ...
  hlq.SUBSCRIBE.FRUIT ...

ICH408I USER(USER2   ) ...
  hlq.SUBSCRIBE.SYSTEM.BASE.TOPIC ...
```

- Multi On Multiplatforms, the following authorization event:

```
MQRC_NOT_AUTHORIZED
ReasonQualifier      MQRQ_SUB_NOT_AUTHORIZED
UserIdentifier        USER2
AdminTopicNames      FRUIT, SYSTEM.BASE.TOPIC
TopicString           "Price/Fruit/Apples"
```

Note the following:

- z/OS The messages you receive on z/OS are identical to those received in the previous task as the same topic objects and profiles are controlling the access.
- Multi The event message you receive on Multiplatforms is similar to the one received in the previous task, but the actual topic string is different.

Grant another user access to subscribe to only the topic deeper within the tree

This topic is the third in a list of tasks that tells you how to grant access to subscribe to topics by more than one user.

Before you begin

This topic uses the setup described in "Grant access to a user to subscribe to a topic deeper within the tree" on page 481.

About this task

In "Grant access to a user to subscribe to a topic deeper within the tree" on page 481, USER2 was refused access to topic `"Price/Fruit/Apples"`. This topic tells you how to grant access to that topic, but not to any other topics.

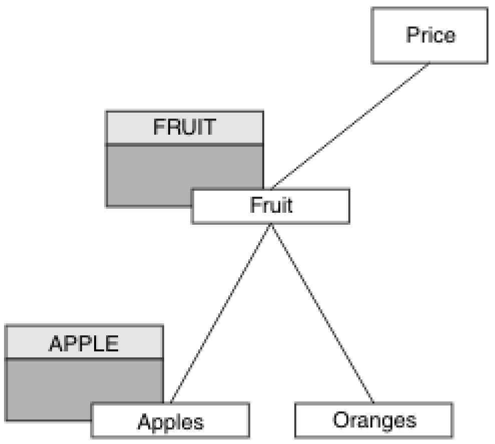


Figure 25. Granting access to specific topics within a topic tree

Table 91. Access requirements for example topics and topic objects

Topic	Subscribe access required	Topic object
Price	No user	None
Price/Fruit	USER1	FRUIT
Price/Fruit/Apples	USER1 and USER2	APPLE
Price/Fruit/Oranges	USER1	

Define a new topic object as follows:

Procedure

1. Issue the MQSC command `DEF TOPIC(APPLE) TOPICSTR('Price/Fruit/Apples')`.

2. Grant access as follows:

   - z/OS **z/OS** :

     In "Grant access to a user to subscribe to a topic deeper within the tree" on page 481 USER1 was granted access to subscribe to topic "Price/Fruit/Apples" by granting the user access to the hlq.SUBSCRIBE.FRUIT profile.

     This single profile also granted USER1 access to subscribe to "Price/Fruit/Oranges" "Price/Fruit/#" and this access remains even with the addition of the new topic object and the profiles associated with it.

     Grant access to USER2 to subscribe to topic "Price/Fruit/Apples" by granting the user access to the hlq.SUBSCRIBE.APPLE profile. Do this, using the following RACF commands:

     ```
     RDEFINE MXTOPIC hlq.SUBSCRIBE.APPLE UACC(NONE)
     PERMIT hlq.SUBSCRIBE.FRUIT APPLE(MXTOPIC) ID(USER2) ACCESS(ALTER)
     ```

   - Multi Multiplatforms:

     In "Grant access to a user to subscribe to a topic deeper within the tree" on page 481 USER1 was granted access to subscribe to topic "Price/Fruit/Apples" by granting the user subscribe access to the FRUIT profile.

     This single profile also granted USER1 access to subscribe to "Price/Fruit/Oranges" and "Price/Fruit/#", and this access remains even with the addition of the new topic object and the profiles associated with it.

     Grant access to USER2 to subscribe to topic "Price/Fruit/Apples" by granting the user subscribe access to the APPLE profile. Do this, using the authorization command for the platform:

     ALW **AIX, Linux, and Windows systems**

     ```
     setmqaut -t topic -n APPLE -p USER2 +sub
     ```

     IBM i **IBM i**

     ```
     GRTMQAUT OBJ(APPLE) OBJTYPE(*TOPIC) USER(USER2) AUT(*SUB)
     ```

Results

z/OS On z/OS, when USER1 attempts to subscribe to topic "Price/Fruit/Apples" the first security check on the hlq.SUBSCRIBE.APPLE profile fails, but on moving up the tree the hlq.SUBSCRIBE.FRUIT profile allows USER1 to subscribe, so the subscription succeeds and no return code is sent to the MQSUB call. However, a RACF ICH message is generated for the first check:

```
ICH408I USER(USER1   ) ...
  hlq.SUBSCRIBE.APPLE ...
```

When USER2 attempts to subscribe to topic "Price/Fruit/Apples" the result is success because the security check passes on the first profile.

When USER2 attempts to subscribe to topic "Price/Fruit/Oranges" the result is failure with an MQRC_NOT_AUTHORIZED message, together with:

- z/OS On z/OS, the following messages seen on the console that show the full security path through the topic tree that has been attempted:

  ```
  ICH408I USER(USER2   ) ...
  ```

```
  hlq.SUBSCRIBE.FRUIT ...

ICH408I USER(USER2   ) ...
  hlq.SUBSCRIBE.SYSTEM.BASE.TOPIC ...
```

- ALW On AIX, Linux, and Windows platforms, the following authorization event:

```
MQRC_NOT_AUTHORIZED
ReasonQualifier      MQRQ_SUB_NOT_AUTHORIZED
UserIdentifier        USER2
AdminTopicNames      FRUIT, SYSTEM.BASE.TOPIC
TopicString           "Price/Fruit/Oranges"
```

- IBM i On IBMi, the following authorization event:

```
MQRC_NOT_AUTHORIZED
ReasonQualifier      MQRQ_SUB_NOT_AUTHORIZED
UserIdentifier        USER2
AdminTopicNames      FRUIT, SYSTEM.BASE.TOPIC
TopicString           "Price/Fruit/Oranges"
```

z/OS The disadvantage of this setup is that, on z/OS, you receive additional `ICH` messages on the console. You can avoid this if you secure the topic tree in a different manner.

Change access control to avoid additional messages

This topic is the fourth in a list of tasks that tells you how to grant access to subscribe to topics by more than one user and to avoid additional RACF ICH408I messages on z/OS.

Before you begin

This topic enhances the setup described in "Grant another user access to subscribe to only the topic deeper within the tree" on page 483 so that you avoid additional error messages.

About this task

This topic tells you how to grant access to topics deeper in the tree, and how to remove access to the topic lower down the tree when no user requires it.

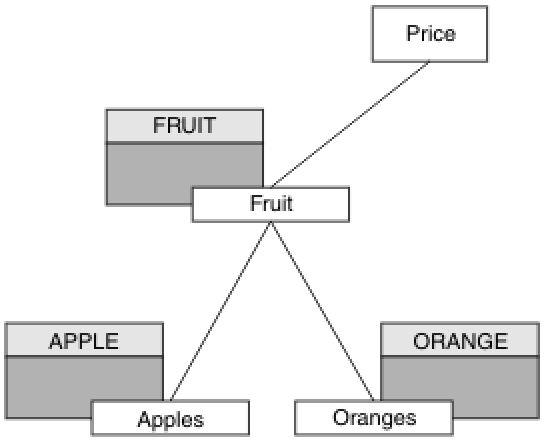


Figure 26. Example of granting access control to avoid additional messages.

Define a new topic object as follows:

Procedure

1. Issue the MQSC command `DEF TOPIC(ORANGE) TOPICSTR('Price/Fruit/Oranges')`.

2. Grant access as follows:

   - z/OS **z/OS** :

     Define a new profile and add access to that profile, and the existing profiles. Do this, using the following RACF commands:

     ```
     RDEFINE MXTOPIC hlq.SUBSCRIBE.ORANGE UACC(NONE)
     PERMIT hlq.SUBSCRIBE.ORANGE CLASS(MXTOPIC) ID(USER1) ACCESS(ALTER)
     PERMIT hlq.SUBSCRIBE.APPLE CLASS(MXTOPIC) ID(USER1) ACCESS(ALTER)
     ```

   - Multi Multiplatforms:

     Set up the equivalent access by using the authorization commands for the platform:

     ALW **AIX, Linux, and Windows systems**

     ```
     setmqaut -t topic -n ORANGE -p USER1 +sub
     setmqaut -t topic -n APPLE -p USER1 +sub
     ```

     IBM i **IBM i**

     ```
     GRTMQAUT OBJ(ORANGE) OBJTYPE(*TOPIC) USER(USER1) AUT(*SUB)
     GRTMQAUT OBJ(APPLE) OBJTYPE(*TOPIC) USER(USER1) AUT(*SUB)
     ```

Results

z/OS On z/OS, when USER1 attempts to subscribe to topic "`Price/Fruit/Apples`" the first security check on the `hlq.SUBSCRIBE.APPLE` profile succeeds.

When USER2 attempts to subscribe to topic "`Price/Fruit/Apples`" the result is success because the security check passes on the first profile.

When USER2 attempts to subscribe to topic "`Price/Fruit/Oranges`" the result is failure with an `MQRC_NOT_AUTHORIZED` message, together with:

- z/OS On z/OS, the following messages seen on the console that show the full security path through the topic tree that has been attempted:

  ```
  ICH408I USER(USER2   ) ...
    hlq.SUBSCRIBE.ORANGE ...

  ICH408I USER(USER2   ) ...
    hlq.SUBSCRIBE.FRUIT ...

  ICH408I USER(USER2   ) ...
    hlq.SUBSCRIBE.SYSTEM.BASE.TOPIC ...
  ```

- ALW On AIX, Linux, and Windows, the following authorization event:

  ```
  MQRC_NOT_AUTHORIZED
  ReasonQualifier      MQRQ_SUB_NOT_AUTHORIZED
  UserIdentifier        USER2
  AdminTopicNames      ORANGE, FRUIT, SYSTEM.BASE.TOPIC
  TopicString           "Price/Fruit/Oranges"
  ```

- IBM i On IBM i, the following authorization event:

  ```
  MQRC_NOT_AUTHORIZED
  ReasonQualifier      MQRQ_SUB_NOT_AUTHORIZED
  UserIdentifier        USER2
  ```

```
AdminTopicNames      ORANGE, FRUIT, SYSTEM.BASE.TOPIC
TopicString           "Price/Fruit/Oranges"
```

Grant access to a user to publish to a topic

This topic is the first one in a list of tasks that tells you how to grant access to publish topics by more than one user.

About this task

This task assumes that no administrative topic objects exist on the right hand side of the topic tree, nor have any profiles been defined for publication. The assumption used is that publishers are using the topic string only.

An application can publish to a topic by providing a topic object, or a topic string, or a combination of both. Whichever way the application selects, the effect is to publish at a certain point in the topic tree. If this point in the topic tree is represented by an administrative topic object, a security profile is checked based on the name of that topic object. For example:

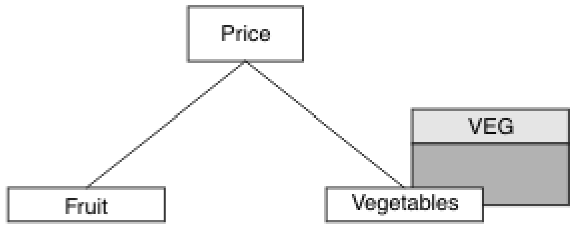


Figure 27. Granting publish access to a topic

Table 92. Example publish access requirements

Topic	Publish access required	Topic object
Price	No user	None
Price/Vegetables	USER1	VEG

Define a new topic object as follows:

Procedure

1. Issue the MQSC command `DEF TOPIC(VEG) TOPICSTR('Price/Vegetables')`.
2. Grant access as follows:

   - z/OS **z/OS** :

     Grant access to USER1 to publish to topic "`Price/Vegetables`" by granting the user access to the `hlq.PUBLISH.VEG` profile. Do this, using the following RACF commands:

     ```
     RDEFINE MXTOPIC hlq.PUBLISH.VEG UACC(NONE)
     PERMIT hlq.PUBLISH.VEG CLASS(MXTOPIC) ID(USER1) ACCESS(UPDATE)
     ```

   - Other platforms:

     Grant access to USER1 to publish to topic "`Price/Vegetables`" by granting the user access to the VEG profile. Do this, using the authorization command for the platform:

     ALW **AIX, Linux, and Windows systems**

     ```
     setmqaut -t topic -n VEG -p USER1 +pub
     ```

IBM i IBM i

```
GRTMQAUT OBJ(VEG) OBJTYPE(*TOPIC) USER(USER1) AUT(*PUB)
```

Results

When USER1 attempts to publish to topic "`Price/Vegetables`" the result is success; that is, the MQOPEN call succeeds.

When USER2 attempts to publish to topic "`Price/Vegetables`" the MQOPEN call fails with an `MQRC_NOT_AUTHORIZED` message, together with:

- z/OS On z/OS, the following messages seen on the console that show the full security path through the topic tree that has been attempted:

```
ICH408I USER(USER2   ) ...
  hlq.PUBLISH.VEG ...

ICH408I USER(USER2   ) ...
  hlq.PUBLISH.SYSTEM.BASE.TOPIC ...
```

- ALW On other platforms, the following authorization event:

```
MQRC_NOT_AUTHORIZED
ReasonQualifier      MQRQ_OPEN_NOT_AUTHORIZED
UserIdentifier        USER2
AdminTopicNames      VEG, SYSTEM.BASE.TOPIC
TopicString           "Price/Vegetables"
```

- IBM i On IBMi, the following authorization event:

```
MQRC_NOT_AUTHORIZED
ReasonQualifier      MQRQ_OPEN_NOT_AUTHORIZED
UserIdentifier        USER2
AdminTopicNames      VEG, SYSTEM.BASE.TOPIC
TopicString           "Price/Vegetables"
```

Note that this is an illustration of what you see; not all the fields.

Grant access to a user to publish to a topic deeper within the tree

This topic is the second in a list of tasks that tells you how to grant access to publish to topics by more than one user.

Before you begin

This topic uses the setup described in "Grant access to a user to publish to a topic" on page 487.

About this task

If the point in the topic tree where the application publishes is not represented by an administrative topic object, move up the tree until the closest parent administrative topic object is located. The security profile is checked, based on the name of that topic object.

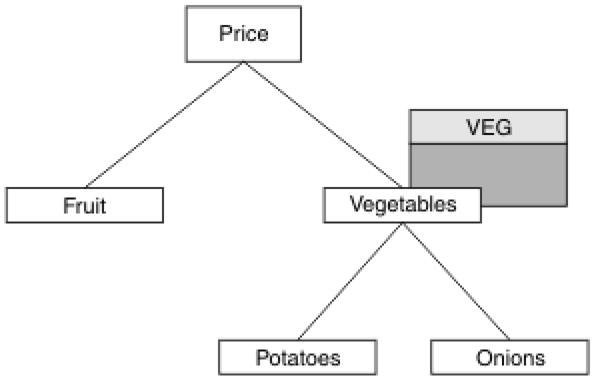


Figure 28. Granting publish access to a topic within a topic tree

Table 93. Example publish access requirements

Topic	Subscribe access required	Topic object
Price	No user	None
Price/Vegetables	USER1	VEG
Price/ Vegetables/ Potatoes	USER1	
Price/ Vegetables/ Onions	USER1	

In the previous task USER1 was granted access to publish topic "`Price/Vegetables/Potatoes`" by granting it access to the `hlq.PUBLISH.VEG` profile on z/OS or publish access to the VEG profile on Multiplatforms. This single profile also grants USER1 access to publish at "`Price/Vegetables/Onions`".

When USER1 attempts to publish at topic "`Price/Vegetables/Potatoes`" the result is success; that is the MQOPEN call succeeds.

When USER2 attempts to subscribe to topic "`Price/Vegetables/Potatoes`" the result is failure; that is, the MQOPEN call fails with an `MQRC_NOT_AUTHORIZED` message, together with:

- z/OS On z/OS, the following messages seen on the console that show the full security path through the topic tree that has been attempted:

```
ICH408I USER(USER2   ) ...
  hlq.PUBLISH.VEG ...

ICH408I USER(USER2   ) ...
  hlq.PUBLISH.SYSTEM.BASE.TOPIC ...
```

- Multi On Multiplatforms, the following authorization event:

```
MQRC_NOT_AUTHORIZED
ReasonQualifier       MQRQ_OPEN_NOT_AUTHORIZED
UserIdentifier         USER2
AdminTopicNames       VEG, SYSTEM.BASE.TOPIC
TopicString            "Price/Vegetables/Potatoes"
```

Note the following:

- z/OS The messages you receive on z/OS are identical to those received in the previous task as the same topic objects and profiles are controlling the access.
- Multi The event message you receive on Multiplatforms is similar to the one received in the previous task, but the actual topic string is different.

Grant access for publish and subscribe

This topic is the last in a list of tasks that tells you how to grant access to publish and subscribe to topics by more than one user.

Before you begin

This topic uses the setup described in "Grant access to a user to publish to a topic deeper within the tree" on page 488.

About this task

In a previous task USER1 was given access to subscribe to the topic `"Price/Fruit"`. This topic tells you how to grant access to that user to publish to that topic.

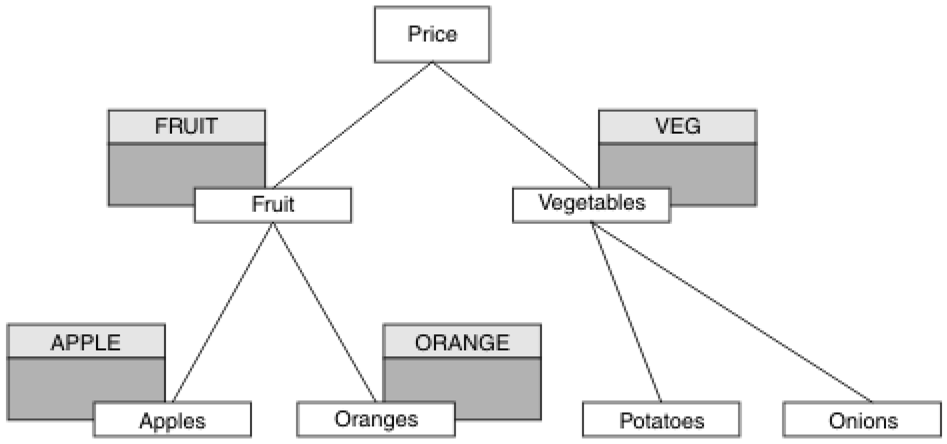


Figure 29. Granting access for publishing and subscribing

Table 94. Example publishing and subscribing access requirements

Topic	Subscribe access required	Publish access required	Topic object
Price	No user	No user	None
Price/Fruit	USER1	USER1	FRUIT
Price/Fruit/ Apples	USER1 and USER2		APPLE
Price/Fruit/ Oranges	USER1		ORANGE

Procedure

Grant access as follows:

- z/OS **z/OS** :

In an earlier task USER1 was granted access to subscribe to topic "Price/Fruit" by granting the user access to the hlq.SUBSCRIBE.FRUIT profile.

In order to publish to the "Price/Fruit" topic, grant access to USER1 to the hlq.PUBLISH.FRUIT profile. Do this, using the following RACF commands:

```
RDEFINE MXTOPIC hlq.PUBLISH.FRUIT UACC(NONE)
PERMIT hlq.PUBLISH.FRUIT CLASS(MXTOPIC) ID(USER1) ACCESS(ALTER)
```

- Multi Multiplatforms:

  Grant access to USER1 to publish to topic "Price/Fruit" by granting the user publish access to the FRUIT profile. Do this, using the authorization command for the platform:

  ALW **AIX, Linux, and Windows systems**

  ```
  setmqaut -t topic -n FRUIT -p USER1 +pub
  ```

  IBM i **IBM i**

  ```
  GRTMQAUT OBJ(FRUIT) OBJTYPE(*TOPIC) USER(USER1) AUT(*PUB)
  ```

Results

z/OS On z/OS, when USER1 attempts to publish to topic "Price/Fruit" the security check on the MQOPEN call passes.

When USER2 attempts to publish at topic "Price/Fruit" the result is failure with an MQRC_NOT_AUTHORIZED message, together with:

- z/OS On z/OS, the following messages seen on the console that show the full security path through the topic tree that has been attempted:

```
ICH408I USER(USER2    ) ...
  hlq.PUBLISH.FRUIT ...

ICH408I USER(USER2    ) ...
  hlq.PUBLISH.SYSTEM.BASE.TOPIC ...
```

- ALW On AIX, Linux, and Windows platforms, the following authorization event:

```
MQRC_NOT_AUTHORIZED
ReasonQualifier      MQRQ_OPEN_NOT_AUTHORIZED
UserIdentifier        USER2
AdminTopicNames      FRUIT, SYSTEM.BASE.TOPIC
TopicString           "Price/Fruit"
```

- IBM i On IBM i, the following authorization event:

```
MQRC_NOT_AUTHORIZED
ReasonQualifier      MQRQ_OPEN_NOT_AUTHORIZED
UserIdentifier        USER2
AdminTopicNames      FRUIT, SYSTEM.BASE.TOPIC
TopicString           "Price/Fruit"
```

Following the complete set of these tasks, gives USER1 and USER2 the following access authorities for publish and subscribe to the topics listed:

Table 95. Complete list of access authorities resulting from security examples			
Topic	**Subscribe access required**	**Publish access required**	**Topic object**
Price	No user	No user	None
Price/Fruit	USER1	USER1	FRUIT
Price/Fruit/ Apples	USER1 and USER2		APPLE
Price/Fruit/ Oranges	USER1		ORANGE
Price/ Vegetables		USER1	VEG
Price/ Vegetables/ Potatoes			
Price/ Vegetables/ Onions			

z/OS Where you have different requirements for security access at different levels within the topic tree, careful planning ensures that you do not receive extraneous security warnings on the z/OS console log. Setting up security at the correct level within the tree avoids misleading security messages.

Subscription security

MQSO_ALTERNATE_USER_AUTHORITY

The AlternateUserId field contains a user identifier to use to validate this MQSUB call. The call can succeed only if this AlternateUserId is authorized to subscribe to the topic with the specified access options, regardless of whether the user identifier under which the application is running is authorized to do so.

MQSO_SET_IDENTITY_CONTEXT

The subscription is to use the accounting token and application identity data supplied in the PubAccountingToken and PubApplIdentityData fields.

If this option is specified, the same authorization check is carried out as if the destination queue was accessed using an MQOPEN call with MQOO_SET_IDENTITY_CONTEXT, except in the case where the MQSO_MANAGED option is also used in which case there is no authorization check on the destination queue.

If this option is not specified, the publications sent to this subscriber have default context information associated with them as follows:

Table 96. Default publication context information	
Field in MQMD	**Value used**
UserIdentifier	The user ID associated with the subscription (see SUBUSER field on DISPLAY SBSTATUS) at the time the publication is made.

Table 96. Default publication context information (continued)

Field in MQMD	Value used
AccountingToken	Determined from the environment if possible; set to MQACT_NONE otherwise.
ApplIdentityData	Set to blanks.

This option is only valid with MQSO_CREATE and MQSO_ALTER. If used with MQSO_RESUME, the PubAccountingToken and PubApplIdentityData fields are ignored, so this option has no effect.

If a subscription is altered without using this option where previously the subscription had supplied identity context information, default context information is generated for the altered subscription.

If a subscription allowing different user IDs to use it with option MQSO_ANY_USERID, is resumed by a different user ID, default identity context is generated for the new user ID now owning the subscription and any subsequent publications are delivered containing the new identity context.

AlternateSecurityId

This is a security identifier that is passed with the AlternateUserId to the authorization service to allow appropriate authorization checks to be performed. AlternateSecurityId is used only if MQSO_ALTERNATE_USER_AUTHORITY is specified, and the AlternateUserId field is not entirely blank up to the first null character or the end of the field.

MQSO_ANY_USERID subscription option

When MQSO_ANY_USERID is specified, the identity of the subscriber is not restricted to a single user ID. This allows any user to alter or resume the subscription when they have suitable authority. Only a single user may have the subscription at any one time. An attempt to resume use of a subscription currently in use by another application will cause the call to fail with MQRC_SUBSCRIPTION_IN_USE.

To add this option to an existing subscription the MQSUB call (using MQSO_ALTER) must come from the same user ID as the original subscription.

If an MQSUB call refers to an existing subscription with MQSO_ANY_USERID set, and the user ID differs from the original subscription, the call succeeds only if the new user ID has authority to subscribe to the topic. After successful completion, future publications to this subscriber are put to the subscriber's queue with the new user ID set in the publication.

MQSO_FIXED_USERID

When MQSO_FIXED_USERID is specified, the subscription can only be altered or resumed by a single owning user ID. This user ID is the last user ID to alter the subscription that set this option, thereby removing the MQSO_ANY_USERID option, or if no alters have taken place, it is the user ID that created the subscription.

If an MQSUB verb refers to an existing subscription with MQSO_ANY_USERID set and alters the subscription (using MQSO_ALTER) to use option MQSO_FIXED_USERID, the user ID of the subscription is now fixed at this new user ID. The call succeeds only if the new user ID has authority to subscribe to the topic.

If a user ID other than the one recorded as owning a subscription trys to resume or alter an MQSO_FIXED_USERID subscription, the call will fail with MQRC_IDENTITY_MISMATCH. The owning user ID of a subscription can be viewed using the DISPLAY SBSTATUS command.

If neither MQSO_ANY_USERID or MQSO_FIXED_USERID is specified, the default is MQSO_FIXED_USERID.

Publish/subscribe security between queue managers

Publish/subscribe internal messages, such as proxy subscriptions and publications, are put to publish/subscribe system queues using normal channel security rules. The information and diagrams in this topic highlight the various processes and user IDs involved in the delivery of these messages.

Local access control

Access to topics for publication and subscriptions is governed by local security definitions and rules that are described in Publish/subscribe security. No local topic object is required to establish access control. Administrators can choose to apply access control to clustered topic objects, irrespective of whether they exist in the cluster yet.

System administrators are responsible for access control on their local system. They must trust the administrators of other members of the hierarchy or cluster collectives to be responsible for their access control policy. Because access control is defined for each separate machine it is likely to be burdensome if fine level control is needed. It might not be necessary to impose any access control, or access control might be defined on high-level objects in the topic tree. Fine level access control can be defined for each subdivision of the topic namespace.

Making a proxy subscription

Trust for an organization to connect its queue manager to your queue manager is confirmed by normal channel authentication means. If that trusted organization is also allowed to do distributed publish/subscribe, an authority check is done. The check is made when the channel puts a message to a distributed publish/subscribe queue. For example, if a message is put to the `SYSTEM.INTER.QMGR.CONTROL` queue. The user ID for the queue authority check depends on the `PUTAUT` values of the receiving channel. For example, the user ID of the channel, `MCAUSER`, the message context, depending on the value and platform. For more information about channel security, see Channel security.

Proxy subscriptions are made with the user ID of the distributed publish/subscribe agent on the remote queue manager. For example, `QM2` in Figure 30 on page 494. The user is then easily granted access to local topic object profiles, because that user ID is defined in the system and there are therefore no domain conflicts.

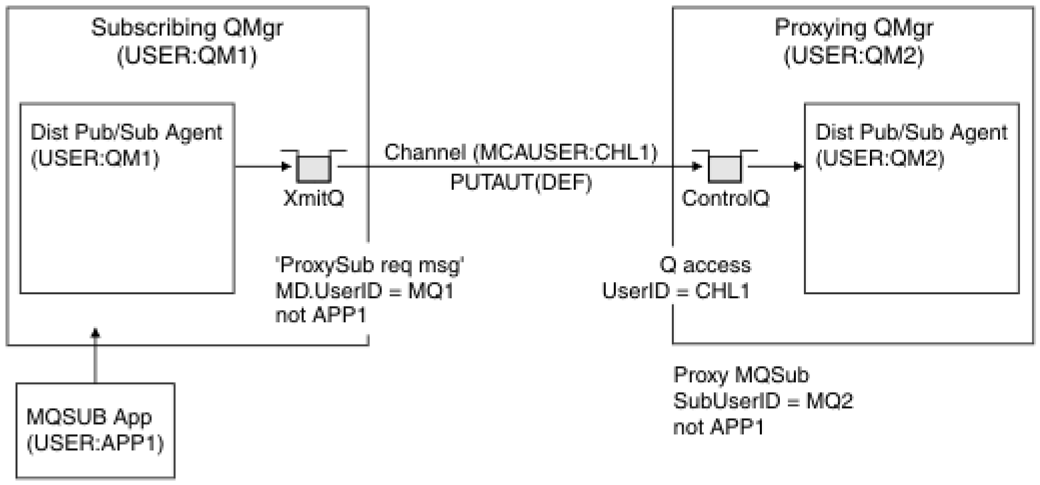


Figure 30. Proxy subscription security, making a subscription

Sending back remote publications

When a publication is created on the publishing queue manager, a copy of the publication is created for any proxy subscription. The context of the copied publication contains the context of the user ID which made the subscription; QM2 in Figure 31 on page 495. The proxy subscription is created with a destination queue that is a remote queue, so the publication message is resolved onto a transmission queue.

Trust for an organization to connect its queue manager, QM2, to another queue manager, QM1, is confirmed by normal channel authentication means. If that trusted organization is then allowed to do distributed publish/subscribe, an authority check is done when the channel puts the publication message to the distributed publish/subscribe publication queue SYSTEM.INTER.QMGR.PUBS. The user ID for the queue authority check depends on the PUTAUT value of the receiving channel (for example, the user ID of the channel, MCAUSER, message context, and others, depending on value and platform). For more information about channel security, see Channel security.

When the publication message reaches the subscribing queue manager, another MQPUT to the topic is done under the authority of that queue manager and the context with the message is replaced by the context of each of the local subscribers as they are each given the message.

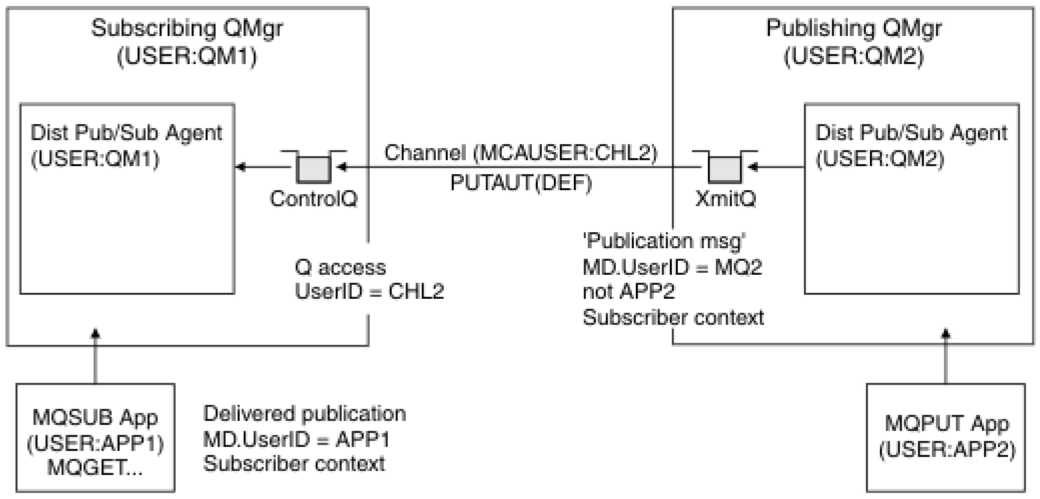


Figure 31. Proxy subscription security, forwarding publications

On a system where little has been considered regarding security, the distributed publish/subscribe processes are likely to be running under a user ID in the mqm group, the MCAUSER parameter on a channel is blank (the default), and messages are delivered to the various system queues as required. The unsecured system makes it easy to set up a proof of concept to demonstrate distributed publish/subscribe.

On a system where security is more seriously considered, these internal messages are subject to the same security controls as any message going over the channel.

If the channel is set up with a non-blank MCAUSER and a PUTAUT value specifying that MCAUSER must be checked, then the MCAUSER in question must be granted access to SYSTEM.INTER.QMGR.* queues. If there are multiple different remote queue managers, with channels running under different MCAUSER IDs, all those user IDs need to be granted access to the SYSTEM.INTER.QMGR.* queues. Channels running under different MCAUSER IDs might occur, for example, when multiple hierarchical connections are configured on a single queue manager.

If the channel is set up with a PUTAUT value specifying that the context of the message is used, then access to the SYSTEM.INTER.QMGR.* queues are checked based on the user ID inside the internal message. Because all these messages are put with the user ID of the distributed publish/subscribe agent from the queue manager that is sending the internal message, or publication message (see Figure 31 on page 495), it is not too large a set of user IDs to grant access to the various system queues (one per remote queue manager), if you want to set up your distributed publish/subscribe security in this way. It still has all the same issues that channel context security always has; that of the different user ID domains and the fact that the user ID in the message might not be defined on the receiving system. However, it is a perfectly acceptable way to run if required.

z/OS System queue security provides a list of queues and the access that is required to securely set up your distributed publish/subscribe environment. If any internal messages or publications fail to be put due to security violations, the channel writes a message to the log in the normal manner and the messages can be sent to the dead-letter queue according to normal channel error processing.

All inter-queue manager messaging for the purposes of distributed publish/subscribe runs using normal channel security.

For information about restricting publications and proxy subscriptions at the topic level, see Publish/subscribe security.

Using default user IDs with a queue manager hierarchy

If you have a hierarchy of queue managers running on different platforms and are using default user IDs, note that these default user IDs differ between platforms and might not be known on the target platform. As a result, a queue manager running on one platform rejects messages received from queue managers on other platforms with the reason code `MQRC_NOT_AUTHORIZED`.

To avoid messages being rejected, at a minimum, the following authorities need to be added to the default user IDs used on other platforms:

- *PUT *GET authority on the SYSTEM.BROKER. queues
- *PUB *SUB authority on the SYSTEM.BROKER. topics
- *ADMCRT *ADMDLT *ADMCHG authority on the SYSTEM.BROKER.CONTROL.QUEUE queue.

The default user IDs with a queue manager hierarchy are as follows:

Platform	Default user ID
Windows	`mqm`
AIX and Linux systems	`mqm`
IBM i	`QMQM`
z/OS	The channel initiator address space user ID

If queue managers on platforms other than IBM i are hierarchically attached to a queue manager on IBM i, create and grant access to the 'qmqm' user ID.

If queue managers on IBM i or z/OS are hierarchically attached to a queue manager on AIX, Linux, and Windows, create and grant access to the 'mqm' user ID.

If queue managers on Multiplatforms are hierarchically attached to a queue manager on z/OS, create and grant access to the z/OS channel initiator address space user ID.

User IDs can be case sensitive. The originating queue manager (if on Multiplatforms) forces the user ID to be all uppercase. The receiving queue manager (if on AIX, Linux, and Windows) forces the user ID to be all lowercase. Therefore, all user IDs created on AIX and Linux systems must be created in their lowercase form. If a message exit has been installed, forcing the user ID into uppercase or lowercase does not take place. Care must be taken to understand how the message exit processes the user ID.

To avoid potential problems with conversion of user IDs:

- On AIX, Linux, and Windows systems, ensure the user IDs are specified in lowercase.
- On IBM i and z/OS systems, ensure the user IDs are specified in uppercase.

IBM MQ Console and REST API security

Security for the IBM MQ Console and the REST API is configured by editing the mqweb server configuration in the `mqwebuser.xml` file.

About this task

You can track user actions and audit the use of the IBM MQ Console and the REST API by examining the log files of the mqweb server.

Users of the IBM MQ Console and the REST API can be authenticated by using:

- Basic registry
- LDAP registry
- Local OS registry
- SAF on z/OS
- Any other registry type supported by WebSphere Liberty

Roles can be assigned to IBM MQ Console users, and to REST API users to determine what level of access they are granted to IBM MQ objects. For example, to perform messaging, users must be assigned the `MQWebUser` role. For more information about the available roles, see "Roles on the IBM MQ Console and REST API" on page 508.

After a user is assigned a role, there are a number of methods that can be used to authenticate the user. With the IBM MQ Console, users can log in with a user name and password, or can use client certificate authentication. With the REST API, users can use basic HTTP authentication, token based authentication, or client certificate authentication.

Procedure

1. Define the user registry to authenticate users, and assign each user or group a role to authorize the users and groups to use the IBM MQ Console or REST API. For more information, see "Configuring users and roles" on page 498
2. Choose how users of the IBM MQ Console authenticate with the mqweb server. You do not have to use the same method for all users:
   - Let users authenticate by using token authentication. In this case, a user enters a user ID and password at the IBM MQ Console log in screen. An LTPA token is generated that enables the user to remain logged in and authorized for a set amount of time. No further configuration is required to use this authentication option, but you can optionally configure the expiry time for the LTPA token. For more information, see Configuring the LTPA token expiry interval.
   - Let users authenticate by using client certificates. In this case, the user does not use a user ID or password to log in to the IBM MQ Console, but uses the client certificate instead. For more information, see "Configuring client certificate authentication with the REST API and IBM MQ Console" on page 512.
3. Choose how users of the REST API authenticate with the mqweb server. You do not have to use the same method for all users:
   - Let users authenticate by using HTTP basic authentication. In this case, a user name and password is encoded, but not encrypted, and sent with each REST API request to authenticate and authorize the user for that request. In order for this authentication to be secure, you must use a secure connection. That is, you must use HTTPS. For more information, see "Using HTTP basic authentication with the REST API" on page 516.
   - Let users authenticate by using token authentication. In this case, a user provides a user ID and password to the REST API `login` resource with the HTTP POST method. An LTPA token is generated that enables the user to remain logged in and authorized for a set amount of time. For more information, see "Using token-based authentication with the REST API " on page 517.

In order for this authentication to be secure, you must use a secure connection. That is, you must use HTTPS. However, if you have enabled HTTP connections, you can allow an LTPA token that is issued for an HTTPS connection to be used for an HTTP connection. For more information, see Configuring the LTPA token.

  - Let users authenticate by using client certificates. In this case, the user does not use a user ID or password to log in to the REST API, but uses the client certificate instead. For more information, see "Configuring client certificate authentication with the REST API and IBM MQ Console" on page 512.

4. Optional: Configure Cross Origin Resource Sharing for the REST API.

   By default, a web browser does not allow scripts, such as JavaScript, to invoke the REST API when the script is not from the same origin as the REST API. That is, cross-origin requests are not enabled. You can configure Cross Origin Resource Sharing (CORS) to allow cross-origin requests from specified URLs. For more information, see "Configuring CORS for the REST API" on page 520.

5. Optional: Configure host header validation for the IBM MQ Console and REST API.

   You can configure host header validation and create an allowlist of hostnames and ports to ensure that only requests that contain specific host headers are processed by the IBM MQ Console and REST API. For more information, see "Configuring host header validation for the IBM MQ Console and REST API" on page 521.

Configuring users and roles

To make use of the IBM MQ Console or the REST API, users need to authenticate against a user registry, defined to the mqweb server.

About this task

Authenticated users need to be a member of one of the groups that authorizes access to the capabilities of the IBM MQ Console and REST API. By default, the user registry does not contain any users; these need to be added by editing the `mqwebuser.xml` file.

When you configure users and groups, you first configure a user registry to authenticate users and groups against. This user registry is shared between the IBM MQ Console and the REST API. You can control whether users and groups have access to the IBM MQ Console, REST API, or both, when you configure roles for your users and groups.

After you configure the user registry, you configure roles for the users and groups to grant them authorization. There are several roles available, including roles specific to using the REST API for Managed File Transfer. Each role grants a different level of access. For more information, see "Roles on the IBM MQ Console and REST API" on page 508.

A number of sample XML files are provided with the mqweb server to make the configuration of users and groups simpler. Users who are familiar with configuring security in WebSphere Liberty (WLP) might prefer not to use the samples. WLP provides other authorization capabilities in addition to the ones documented here.

Procedure

- Configure users and groups with a basic registry by using the `basic_registry.xml` file.

  The user names and passwords in the registry are used to authenticate and authorize users of the IBM MQ Console and the REST API.

  To configure a basic registry by using the `basic_registry.xml` sample file, see "Configuring a basic registry for the IBM MQ Console and REST API" on page 500.

- Configure users and groups with an LDAP registry by using the `ldap_registry.xml` file.

  The user names and passwords in the LDAP registry are used to authenticate and authorize use of the IBM MQ Console and the REST API.

To configure an LDAP registry by using the `ldap_registry.xml` sample file, see "Configuring an LDAP registry for the IBM MQ Console and REST API" on page 504.

- ALW

  Configure users and groups with a local operating system registry by using the `local_os_registry.xml` file.

  The user names and passwords in the operating system registry are used to authenticate and authorize users of the IBM MQ Console and the REST API.

  To configure a local OS registry by using the `local_os_registry.xml` sample file, see "Configuring a local OS registry for the IBM MQ Console and REST API" on page 502.

- z/OS

  Configure users and groups with the System authorization facility (SAF) interface on z/OS by using the `zos_saf_registry.xml` file.

  RACF, or other security product, profiles are used to grant users and groups access to roles. The user names and passwords in the RACF database are used to authenticate and authorize users of the IBM MQ Console and REST API.

  To configure the SAF interface by using the `zos_saf_registry.xml` sample file, see "Configuring a SAF registry for the IBM MQ Console and REST API" on page 506.
- Disable security, including the ability to access the IBM MQ Console, or the REST API, using HTTPS, by using the `no_security.xml` file.

What to do next

Choose how users authenticate:

IBM MQ Console authentication options

- Let users authenticate by using token authentication. In this case, a user enters a user ID and password at the IBM MQ Console log in screen. An LTPA token is generated that enables the user to remain logged in and authorized for a set amount of time. No further configuration is required to use this authentication option, but you can optionally configure the expiry interval for the LTPA token. For more information, see Configuring the LTPA token expiry interval.
- Let users authenticate by using client certificates. In this case, the user does not use a user ID or password to log in to the IBM MQ Console, but uses the client certificate instead. For more information, see "Configuring client certificate authentication with the REST API and IBM MQ Console" on page 512.

REST API authentication options

- Let users authenticate by using HTTP basic authentication. In this case, a user name and password is encoded, but not encrypted, and sent with each REST API request to authenticate and authorize the user for that request. In order for this authentication to be secure, you must use a secure connection. That is, you must use HTTPS. For more information, see "Using HTTP basic authentication with the REST API" on page 516.
- Let users authenticate by using token authentication. In this case, a user provides a user ID and password to the REST API `login` resource with the HTTP POST method. An LTPA token is generated that enables the user to remain logged in and authorized for a set amount of time. For more information, see "Using token-based authentication with the REST API " on page 517. You can configure the expiry interval for the LTPA token. For more information, see Configuring the LTPA token.
- Let users authenticate by using client certificates. In this case, the user does not use a user ID or password to log in to the REST API, but uses the client certificate instead. For more information, see "Configuring client certificate authentication with the REST API and IBM MQ Console" on page 512.

Configuring a basic registry for the IBM MQ Console and REST API

You can configure a basic registry within the `mqwebuser.xml` file. The user names, passwords, and roles in the XML file are used to authenticate and authorize users of the IBM MQ Console and the REST API.

Before you begin

- When you configure users within the basic registry, you must assign each user a role. Each role provides different levels of privilege to access the IBM MQ Console and REST API, and determines the security context that is used when an allowed operation is attempted. You need to understand these roles before you configure the basic registry. For more information about each of the roles, see "Roles on the IBM MQ Console and REST API" on page 508.
- To complete this task, you must be a user with sufficient privileges to edit the `mqwebuser.xml` file:
  - z/OS On z/OS, you must have write access to the `mqwebuser.xml` file.
  - Multi On all other operating systems, you must be a privileged user.
  - Linux V 9.4.0 If the mqweb server is part of a stand-alone IBM MQ Web Server installation, you must have write access to the `mqwebuser.xml` file in the IBM MQ Web Server data directory.

Procedure

1. Copy the sample XML file `basic_registry.xml` from one of the following paths:
   - In an IBM MQ installation:
     - ALW On AIX, Linux, and Windows: *MQ_INSTALLATION_PATH* `/web/mq/samp/configuration`
     - z/OS On z/OS: `PathPrefix /web/mq/samp/configuration`

       where `PathPrefix` is the IBM MQ for z/OS UNIX System Services Components installation path.
   - Linux V 9.4.0 In a stand-alone IBM MQ Web Server installation: *MQWEB_INSTALLATION_PATH*`/web/mq/samp/configuration`

     where *MQWEB_INSTALLATION_PATH* is the directory to which the IBM MQ Web Server installation file was decompressed.
2. Place the sample file in the appropriate directory:
   - In an IBM MQ installation:
     - Linux AIX On AIX or Linux: `/var/mqm/web/installations/`*`installationName`*`/servers/mqweb`
     - Windows On Windows: *MQ_DATA_PATH*`\web\installations\`*`installationName`*`\servers\mqweb`, where *MQ_DATA_PATH* is the IBM MQ data path. This path is the data path that is selected during the installation of IBM MQ. By default, this path is `C:\ProgramData\IBM\MQ`.
     - z/OS On z/OS: *`WLP_user_directory`*`/servers/mqweb`

       where *WLP_user_directory* is the directory that was specified when the **`crtmqweb`** script ran to create the mqweb server definition.
   - Linux V 9.4.0 In a stand-alone IBM MQ Web Server installation: *MQ_OVERRIDE_DATA_PATH*`/web/installations/MQWEBINST/servers/mqweb`

     where *MQ_OVERRIDE_DATA_PATH* is the IBM MQ Web Server data directory that the **`MQ_OVERRIDE_DATA_PATH`** environment variable points to.

3. Optional: If you changed any configuration settings in `mqwebuser.xml`, copy them into the sample file.
4. Delete the existing `mqwebuser.xml` file and rename the sample file to `mqwebuser.xml`.
5. Edit the new `mqwebuser.xml` file to add users and groups within the **`basicRegistry`** tags.

   Be aware that any user with the `MQWebUser` role can perform only the operations that the user ID is granted to perform on the queue manager. Therefore, the user ID defined in the registry must have an identical user ID on the system on which IBM MQ is installed. These user IDs must be in the same case, or the mapping between the user IDs can fail.

   For more information about configuring basic user registries, see Configuring a basic user registry for Liberty in the WebSphere Liberty documentation.
6. Assign roles to users and groups by editing the `mqwebuser.xml` file:

   There are several roles available that authorize users and groups to use the IBM MQ Console, and the REST API. Each role grants a different level of access. For more information, see "Roles on the IBM MQ Console and REST API" on page 508.

   - To assign roles and grant access to the IBM MQ Console, add your users and groups between the appropriate **`security-role`** tags within the **`<enterpriseApplication id="com.ibm.mq.console">`** tags.
   - To assign roles and grant access to the REST API, add your users and groups between the appropriate **`security-role`** tags within the **`<enterpriseApplication id="com.ibm.mq.rest">`** tags.

   For help with the format of the user and group information within the **`security-role`** tags, see the examples.
7. If you provided passwords for users in `mqwebuser.xml`, you should encode these passwords, to make them more secure, by using the **`securityUtility encoding`** command provided by WebSphere Liberty. For more information, see Liberty:securityUtility command in the WebSphere Liberty product documentation.

Example

In the following example, the group `MQWebAdminGroup` is granted access to the IBM MQ Console with the role `MQWebAdmin`. The user, `reader`, is granted access with the role `MQWebAdminRO`, and the user `guest` is granted access with the role `MQWebUser`:

```
<enterpriseApplication id="com.ibm.mq.console">
        <application-bnd>
            <security-role name="MQWebAdmin">
                <group name="MQWebAdminGroup" realm="defaultRealm"/>
            </security-role>
            <security-role name="MQWebAdminRO">
                <user name="reader" realm="defaultRealm"/>
            </security-role>
            <security-role name="MQWebUser">
                <user name="guest" realm="defaultRealm"/>
            </security-role>
        </application-bnd>
    </enterpriseApplication>
```

In the following example, the users `reader` and `guest` are granted access to the IBM MQ Console. The user `user` is granted access to the REST API, and any users within the `MQAdmin` group are granted access to the IBM MQ Console and the REST API. The `mftadmin` user is granted access to the REST API for MFT :

```
<enterpriseApplication id="com.ibm.mq.console">
    <application-bnd>
        <security-role name="MQWebAdmin">
            <group name="MQAdmin" realm="defaultRealm"/>
        </security-role>
        <security-role name="MQWebAdminRO">
            <user name="reader" realm="defaultRealm"/>
        </security-role>
        <security-role name="MQWebUser">
            <user name="guest" realm="defaultRealm"/>
        </security-role>
    </application-bnd>
</enterpriseApplication>

<enterpriseApplication id="com.ibm.mq.rest">
    <application-bnd>
        <security-role name="MQWebAdmin">
            <group name="MQAdmin" realm="defaultRealm"/>
        </security-role>
        <security-role name="MQWebUser">
            <user name="user" realm="defaultRealm"/>
        </security-role>
        <security-role name="MFTWebAdmin">
            <user name="mftadmin" realm="defaultRealm"/>
        </security-role>
     </application-bnd>
</enterpriseApplication>
```

What to do next

Choose how users authenticate:

IBM MQ Console authentication options

- Let users authenticate by using token authentication. In this case, a user enters a user ID and password at the IBM MQ Console log in screen. An LTPA token is generated that enables the user to remain logged in and authorized for a set amount of time. No further configuration is required to use this authentication option, but you can optionally configure the expiry interval for the LTPA token. For more information, see Configuring the LTPA token expiry interval.
- Let users authenticate by using client certificates. In this case, the user does not use a user ID or password to log in to the IBM MQ Console, but uses the client certificate instead. For more information, see "Configuring client certificate authentication with the REST API and IBM MQ Console" on page 512.

REST API authentication options

- Let users authenticate by using HTTP basic authentication. In this case, a user name and password is encoded, but not encrypted, and sent with each REST API request to authenticate and authorize the user for that request. In order for this authentication to be secure, you must use a secure connection. That is, you must use HTTPS. For more information, see "Using HTTP basic authentication with the REST API" on page 516.
- Let users authenticate by using token authentication. In this case, a user provides a user ID and password to the REST API `login` resource with the HTTP POST method. An LTPA token is generated that enables the user to remain logged in and authorized for a set amount of time. For more information, see "Using token-based authentication with the REST API " on page 517. You can configure the expiry interval for the LTPA token. For more information, see Configuring the LTPA token.
- Let users authenticate by using client certificates. In this case, the user does not use a user ID or password to log in to the REST API, but uses the client certificate instead. For more information, see "Configuring client certificate authentication with the REST API and IBM MQ Console" on page 512.

ALW Configuring a local OS registry for the IBM MQ Console and REST API

You can configure a local operating system registry within the `mqwebuser.xml` file. The user names and passwords on the local operating system are used to authenticate and authorize users of the IBM MQ Console and the REST API.

Before you begin

- For client certificate authentication with the local OS authentication feature, the user identity is the common name (CN) from the distinguished name (DN) of the client certificate. If the user identity does not exist as an operating system user, client certificate login will fail and fallback to password based authentication.

- To complete this task, you must be a user with sufficient privileges to edit the `mqwebuser.xml` file:
  - Linux V 9.4.0 If the mqweb server is part of a stand-alone IBM MQ Web Server installation, you must have write access to the `mqwebuser.xml` file in the IBM MQ Web Server data directory.
  - If the mqweb server is part of an IBM MQ installation, you must be a privileged user.

About this task

With a local operating system registry, users and groups are automatically assigned a role:

- Any user that is part of the 'mqm' group, or the 'QMQMADM' group on IBM i, is granted the `MQWebAdmin` and `MFTWebAdmin` roles.
- All other users are granted the `MQWebUser` role.

For more information about these roles, see "Roles on the IBM MQ Console and REST API" on page 508.

A local operating system registry can only be used on AIX, Linux, and Windows. z/OS Equivalent function is provided on z/OS by configuring a SAF registry. For more information, see "Configuring a SAF registry for the IBM MQ Console and REST API" on page 506.

Procedure

1. Copy the sample XML file `local_os_registry.xml` from one of the following paths:
   - Linux V 9.4.0 In a stand-alone IBM MQ Web Server installation: *MQWEB_INSTALLATION_PATH*`/web/mq/samp/configuration`

     where *MQWEB_INSTALLATION_PATH* is the directory to which the IBM MQ Web Server installation file was decompressed.
   - In an IBM MQ installation: *MQ_INSTALLATION_PATH*`/web/mq/samp/configuration`
2. Place the sample file in one of the following directories:
   - Linux V 9.4.0 In a stand-alone IBM MQ Web Server installation: *MQ_OVERRIDE_DATA_PATH*`/web/installations/MQWEBINST/servers/mqweb`

     where *MQ_OVERRIDE_DATA_PATH* is the IBM MQ Web Server data directory that the **`MQ_OVERRIDE_DATA_PATH`** environment variable points to.
   - In an IBM MQ installation: *MQ_DATA_PATH*`/web/installations/`*`installationName`*`/servers/mqweb`
3. Optional: If you changed any configuration settings in `mqwebuser.xml`, copy them into the sample file.
4. Delete the existing `mqwebuser.xml` file and rename the sample file to `mqwebuser.xml`.

What to do next

Choose how users authenticate:

IBM MQ Console authentication options

- Let users authenticate by using token authentication. In this case, a user enters a user ID and password at the IBM MQ Console log in screen. An LTPA token is generated that enables the user to remain logged in and authorized for a set amount of time. No further configuration is required to use this authentication option, but you can optionally configure the expiry interval for the LTPA token. For more information, see Configuring the LTPA token expiry interval.

- Let users authenticate by using client certificates. In this case, the user does not use a user ID or password to log in to the IBM MQ Console, but uses the client certificate instead. For more information, see "Configuring client certificate authentication with the REST API and IBM MQ Console" on page 512.

REST API authentication options

- Let users authenticate by using HTTP basic authentication. In this case, a user name and password is encoded, but not encrypted, and sent with each REST API request to authenticate and authorize the user for that request. In order for this authentication to be secure, you must use a secure connection. That is, you must use HTTPS. For more information, see "Using HTTP basic authentication with the REST API" on page 516.
- Let users authenticate by using token authentication. In this case, a user provides a user ID and password to the REST API `login` resource with the HTTP POST method. An LTPA token is generated that enables the user to remain logged in and authorized for a set amount of time. For more information, see "Using token-based authentication with the REST API " on page 517. You can configure the expiry interval for the LTPA token. For more information, see Configuring the LTPA token.
- Let users authenticate by using client certificates. In this case, the user does not use a user ID or password to log in to the REST API, but uses the client certificate instead. For more information, see "Configuring client certificate authentication with the REST API and IBM MQ Console" on page 512.

Configuring an LDAP registry for the IBM MQ Console and REST API

You can configure an LDAP registry within the `mqwebuser.xml` file. The user names and passwords in the LDAP registry are used to authenticate and authorize users of the IBM MQ Console and the REST API.

Before you begin

- When you configure an LDAP registry, you must assign each user a role. Each role provides different levels of privilege to access the IBM MQ Console and REST API, and determines the security context that is used when an allowed operation is attempted. You need to understand these roles before you configure the registry. For more information about each of the roles, see "Roles on the IBM MQ Console and REST API" on page 508.

  Be aware that any user with the `MQWebUser` role can perform only the operations that the user ID is granted to perform on the queue manager. Therefore, the user ID defined on the LDAP server must have an identical user ID on the system on which IBM MQ is installed. These user IDs must be in the same case, or the mapping between the user IDs can fail.
- To complete this task, you must be a user with sufficient privileges to edit the `mqwebuser.xml` file:
  - z/OS On z/OS, you must have write access to the `mqwebuser.xml` file.
  - Multi On all other operating systems, you must be a privileged user.
  - Linux V 9.4.0 If the mqweb server is part of a stand-alone IBM MQ Web Server installation, you must have write access to the `mqwebuser.xml` file in the IBM MQ Web Server data directory.

Procedure

1. Copy the sample XML file `ldap_registry.xml` from one of the following paths:
   - In an IBM MQ installation:
     - ALW On AIX, Linux, and Windows: *MQ_INSTALLATION_PATH* `/web/mq/samp/configuration`
     - z/OS On z/OS: `PathPrefix /web/mq/samp/configuration`

where `PathPrefix` is the IBM MQ for z/OS UNIX System Services Components installation path.

- Linux V 9.4.0 In a stand-alone IBM MQ Web Server installation: `*MQWEB_INSTALLATION_PATH*/web/mq/samp/configuration`

  where *MQWEB_INSTALLATION_PATH* is the directory to which the IBM MQ Web Server installation file was decompressed.

2. Place the sample file in the appropriate directory:
   - In an IBM MQ installation:
     - Linux AIX On AIX or Linux: `/var/mqm/web/installations/*installationName*/servers/mqweb`
     - Windows On Windows: `*MQ_DATA_PATH*\web\installations\*installationName*\servers\mqweb`, where *MQ_DATA_PATH* is the IBM MQ data path. This path is the data path that is selected during the installation of IBM MQ. By default, this path is `C:\ProgramData\IBM\MQ`.
     - z/OS On z/OS: `*WLP_user_directory*/servers/mqweb`

       where *WLP_user_directory* is the directory that was specified when the **crtmqweb** script ran to create the mqweb server definition.
   - Linux V 9.4.0 In a stand-alone IBM MQ Web Server installation: `*MQ_OVERRIDE_DATA_PATH*/web/installations/MQWEBINST/servers/mqweb`

     where *MQ_OVERRIDE_DATA_PATH* is the IBM MQ Web Server data directory that the **MQ_OVERRIDE_DATA_PATH** environment variable points to.
3. Optional: If you changed any configuration settings in `mqwebuser.xml`, copy them into the sample file.
4. Delete the existing `mqwebuser.xml` file and rename the sample file to `mqwebuser.xml`.
5. Edit the new `mqwebuser.xml` file to change the LDAP registry settings within the **ldapRegistry** and **idsLdapFilterProperties** tags.

   For more information about configuring LDAP registries, see Configuring LDAP user registries in Liberty in the WebSphere Liberty documentation.
6. Assign roles to users and groups by editing the `mqwebuser.xml` file:

   There are several roles available that authorize users and groups to use the IBM MQ Console, and the REST API. Each role grants a different level of access. For more information, see "Roles on the IBM MQ Console and REST API" on page 508.
   - To assign roles and grant access to the IBM MQ Console, add your users and groups between the appropriate **security-role** tags within the **<enterpriseApplication id="com.ibm.mq.console">** tags.
   - To assign roles and grant access to the REST API, add your users and groups between the appropriate **security-role** tags within the **<enterpriseApplication id="com.ibm.mq.rest">** tags.

What to do next

Choose how users authenticate:

IBM MQ Console authentication options

- Let users authenticate by using token authentication. In this case, a user enters a user ID and password at the IBM MQ Console log in screen. An LTPA token is generated that enables the user to remain logged in and authorized for a set amount of time. No further configuration is required to use this authentication option, but you can optionally configure the expiry interval for the LTPA token. For more information, see Configuring the LTPA token expiry interval.

- Let users authenticate by using client certificates. In this case, the user does not use a user ID or password to log in to the IBM MQ Console, but uses the client certificate instead. For more information, see "Configuring client certificate authentication with the REST API and IBM MQ Console" on page 512.

REST API authentication options

- Let users authenticate by using HTTP basic authentication. In this case, a user name and password is encoded, but not encrypted, and sent with each REST API request to authenticate and authorize the user for that request. In order for this authentication to be secure, you must use a secure connection. That is, you must use HTTPS. For more information, see "Using HTTP basic authentication with the REST API" on page 516.
- Let users authenticate by using token authentication. In this case, a user provides a user ID and password to the REST API `login` resource with the HTTP POST method. An LTPA token is generated that enables the user to remain logged in and authorized for a set amount of time. For more information, see "Using token-based authentication with the REST API " on page 517. You can configure the expiry interval for the LTPA token. For more information, see Configuring the LTPA token.
- Let users authenticate by using client certificates. In this case, the user does not use a user ID or password to log in to the REST API, but uses the client certificate instead. For more information, see "Configuring client certificate authentication with the REST API and IBM MQ Console" on page 512.

z/OS Configuring a SAF registry for the IBM MQ Console and REST API

The System Authorization Facility (SAF) interface allows the mqweb server to call the external security manager for authentication and authorization checking. A user can then log in to the IBM MQ Console and REST API with a z/OS user ID and password.

Before you begin

- When you configure a SAF registry, you must assign users a role. Each role provides different levels of privilege to access the IBM MQ Console and REST API, and determines the security context that is used when an allowed operation is attempted. You need to understand these roles before you configure the registry. For more information about each of the roles, see "Roles on the IBM MQ Console and REST API" on page 508.
- You need the WebSphere Liberty Angel process running to use the authorized interface to SAF. See Enabling z/OS authorized services on Liberty for z/OS for more information.
- To complete this task, you must have write access to the `mqwebuser.xml` file, and authority to define security manager profiles.

Note: From IBM MQ 9.3.5 for Continuous Delivery and from IBM MQ 9.3.0 Fix Pack 20 for Long Term Support, the sample configuration file `zos_saf_registry.xml` is updated to remove a duplicate safAuthorization entry.

This update fixes an issue where an ICH408I error can occur when the IBM MQ Console on z/OS is upgraded to a level that ships WebSphere Liberty Profile 22.0.0.12 or later: that is, from IBM MQ 9.3.0 Fix Pack 2 for Long Term Support and from IBM MQ 9.3.1 CSU 1 and IBM MQ 9.3.2 for Continuous Delivery. Having more than one safAuthorization statement is not supported and might cause an ICH408I error when users who are not in either MQWebAdmin or MQWebAdminRO roles, in the EBJROLE class, try to access a z/OS queue manager through the IBM MQ Console.

The default for **`racRouteLog`**, which specifies the types of access attempts to log, is `NONE`. If you require an additional report or record for security auditing, see SAF Authorization (safAuthorization) for more information.

About this task

The SAF interface allows the mqweb server to call the external security manager for authentication and authorization checking for both the IBM MQ Console and REST API.

Procedure

1. Follow the steps in Enabling z/OS authorized services on Liberty for z/OS to give your mqweb server access to use z/OS authorized services.

   Sample JCL for starting the angel process is in `USS_ROOT/web/templates/zos/procs/bbgzangl.jcl`, where USS_ROOT is the path in z/OS UNIX System Services (z/OS UNIX) where z/OS UNIX components are installed.

   In `bbgzangl.jcl`, change the SET ROOT statement to point to `USS_ROOT/web`, for example, `/usr/lpp/mqm/V9R2M0/web`.

   See Administering Liberty on z/OS for further information on stopping and starting the angel process.
2. Follow the steps in Liberty: Setting up the System Authorization Facility (SAF) unauthenticated user to create the unauthenticated user needed by Liberty.
3. Copy the `zos_saf_registry.xml` file from the following path: `PathPrefix /web/mq/samp/configuration` where `PathPrefix` is the z/OS UNIX Components installation path.
4. Place the sample file in the *`WLP_user_directory`*`/servers/mqweb` directory, where *WLP_user_directory* is the directory that was specified when the **`crtmqweb`** script ran to create the mqweb server definition.
5. Optional: If you previously changed any configuration settings in `mqwebuser.xml`, copy them into the sample file.
6. Delete the existing `mqwebuser.xml` file and rename the sample file to `mqwebuser.xml`.
7. Customize the **`safCredentials`** element in `mqwebuser.xml`.

   a. Set **`profilePrefix`** to a name that is unique to your Liberty server. If you have more than one mqweb server running on a single system, you will need to choose a different name for each server; for example MQWEB920 and MQWEB915.

   b. Set **`unauthenticatedUser`** to the name of the unauthenticated user created in step “2” on page 507.
8. Define the mqweb server APPLID to RACF.

   The APPLID resource name is the value you specified in the **`profilePrefix`** attribute in step “7” on page 507. The following example defines the mqweb server APPLID in RACF:

   ```
   RDEFINE APPL profilePrefix UACC(NONE)
   ```
9. Grant all users, or groups, to be authenticated to the IBM MQ Console or REST API READ access to the mqweb server APPLID in the APPL class.

   You must also do this for the unauthenticated user defined in step “2” on page 507. The following example grants a user READ access to the mqweb server APPLID in RACF:

   ```
   PERMIT profilePrefix CLASS(APPL) ACCESS(READ) ID(userID)
   ```
10. Use the **SETROPTS** RACF command to refresh the in-storage RACLISTed APPL class profiles:

    `SETROPTS RACLIST(APPL) REFRESH`
11. Define the profiles in the EJBROLE class needed to give users access to roles in the IBM MQ Console and REST API.

    The following example defines the profiles in RACF, where **`profilePrefix`** is the value specified for the **`profilePrefix`** attribute in step “7” on page 507.

```
RDEFINE EJBROLE profilePrefix.com.ibm.mq.console.MQWebAdmin UACC(NONE)
RDEFINE EJBROLE profilePrefix.com.ibm.mq.console.MQWebAdminRO UACC(NONE)
RDEFINE EJBROLE profilePrefix.com.ibm.mq.console.MQWebUser UACC(NONE)
RDEFINE EJBROLE profilePrefix.com.ibm.mq.rest.MQWebAdmin UACC(NONE)
RDEFINE EJBROLE profilePrefix.com.ibm.mq.rest.MQWebAdminRO UACC(NONE)
RDEFINE EJBROLE profilePrefix.com.ibm.mq.rest.MQWebUser UACC(NONE)
```

```
RDEFINE EJBROLE profilePrefix.com.ibm.mq.rest.MFTWebAdmin UACC(NONE)
RDEFINE EJBROLE profilePrefix.com.ibm.mq.rest.MFTWebAdminRO UACC(NONE)
```

12. Grant users access to roles in the IBM MQ Console and REST API.

    To do this, give users or groups READ access to one or more of the profiles in the EBJROLE class created in step "11" on page 507. For more information about the roles, see "Roles on the IBM MQ Console and REST API" on page 508.

    The following example gives a user access to the MQWebAdmin role for the REST API in RACF, where **profilePrefix** is the value specified for the **profilePrefix** attribute in step "7" on page 507.

```
PERMIT profilePrefix.com.ibm.mq.rest.MQWebAdmin CLASS(EJBROLE) ACCESS(READ) ID(userID)
```

Results

You have set up SAF authentication for the IBM MQ Console and REST API.

What to do next

Choose how users authenticate:

IBM MQ Console authentication options

- Let users authenticate by using token authentication. In this case, a user enters a user ID and password at the IBM MQ Console log in screen. An LTPA token is generated that enables the user to remain logged in and authorized for a set amount of time. No further configuration is required to use this authentication option, but you can optionally configure the expiry interval for the LTPA token. For more information, see Configuring the LTPA token expiry interval.
- Let users authenticate by using client certificates. In this case, the user does not use a user ID or password to log in to the IBM MQ Console, but uses the client certificate instead. For more information, see "Configuring client certificate authentication with the REST API and IBM MQ Console" on page 512.

REST API authentication options

- Let users authenticate by using HTTP basic authentication. In this case, a user name and password is encoded, but not encrypted, and sent with each REST API request to authenticate and authorize the user for that request. In order for this authentication to be secure, you must use a secure connection. That is, you must use HTTPS. For more information, see "Using HTTP basic authentication with the REST API" on page 516.
- Let users authenticate by using token authentication. In this case, a user provides a user ID and password to the REST API `login` resource with the HTTP POST method. An LTPA token is generated that enables the user to remain logged in and authorized for a set amount of time. For more information, see "Using token-based authentication with the REST API " on page 517. You can configure the expiry interval for the LTPA token. For more information, see Configuring the LTPA token.
- Let users authenticate by using client certificates. In this case, the user does not use a user ID or password to log in to the REST API, but uses the client certificate instead. For more information, see "Configuring client certificate authentication with the REST API and IBM MQ Console" on page 512.

Roles on the IBM MQ Console and REST API

When you authorize users and groups to use the IBM MQ Console or REST API, you must assign the users and groups one of the available roles: **MQWebAdmin**, **MQWebAdminRO**, **MQWebUser**, **MFTWebAdmin**, and **MFTWebAdminRO**. Each role provides different levels of privilege to access the IBM MQ Console and REST API, and determines the security context that is used when an allowed operation is attempted.

Note: With the exception of the **MQWebUser** role, the user ID is not case sensitive. See "MQWebUser" on page 509 for the specific requirements for this role.

MQWebAdmin

A user or group that is assigned this role can perform all administrative operations, and operates under the security context of the operating system user ID that is used to start the mqweb server.

A user or group with this role does not have access to the following REST services:

- The REST API for MFT. To use these services, the user or group must also be assigned the **MFTWebAdmin** or **MFTWebAdminRO** role.
- The messaging REST API. To use the messaging REST API, the user must be assigned the **MQWebUser** role.

MQWebAdminRO

This role gives read only access to the IBM MQ Console or REST API. A user or group that is assigned this role can perform the following operations:

- Display and inquire operations on IBM MQ objects such as queues and channels.
- Browse messages on queues.

A user or group that is assigned this role operates under the security context of the operating system user ID that is used to start the mqweb server.

A user or group with this role does not have access to the following REST services:

- The REST API for MFT. To use these services, the user or group must also be assigned the **MFTWebAdmin** or **MFTWebAdminRO** role.
- The messaging REST API. To use the messaging REST API, the user must be assigned the **MQWebUser** role.

MQWebUser

A user or group that is assigned this role can perform any operation that the user ID is granted to perform on the queue manager. For example:

- Start and stop operations on IBM MQ objects such as channels.
- Define and set operations on IBM MQ objects such as queues and channels.
- Display and inquire operations on IBM MQ objects such as queues and channels.
- Put and get messages using the messaging REST API.

A user or group that is assigned this role operates under the security context of the principal, and can perform only the operations that the user ID is granted to perform on the queue manager.

Therefore, the user or group that is defined in the mqweb user registry must be given authority within IBM MQ before that user can perform any operations. By using this role, you can finely control which users have which type of access to specific IBM MQ resources when they use the IBM MQ Console and REST API.

Note:

- The maximum length of a user ID that is assigned this role is 12 characters.
- The case of the user ID must be the same in the mqweb user registry and on the IBM MQ system. If the case of the user ID is different, the user might be authenticated by the IBM MQ Console and REST API but not authorized to use IBM MQ resources.

MFTWebAdmin

A user or group assigned this role can perform all MFT REST operations, and operates under the security context of the operating system user ID that is used to start the mqweb server.

A user or group with this role does not have access to any of the IBM MQ REST API services. To use these services, the user or group must also be assigned the **MQWebAdmin**, **MQWebAdminRO**, or **MQWebUser** role.

MFTWebAdminRO

This role gives read only access to the REST API for MFT . A user or group that is assigned this role can perform read only operations (GET requests) like list transfer and list agents.

A user or group that is assigned this role operates under the security context of the operating system user ID that is used to start the mqweb server.

A user or group with this role does not have access to any of the IBM MQ REST API services. To use these services, the user or group must also be assigned the **MQWebAdmin**, **MQWebAdminRO**, or **MQWebUser** role.

For more information about configuring users and groups to use these roles, see "Configuring users and roles" on page 498.

Overlapping roles

A user or group can be assigned more than one role. When a user performs an operation in this situation, the highest privilege role that is applicable to the operation is used. For example, if a user with the roles **MQWebAdminRO** and **MQWebUser** performs an inquire queue operation, the **MQWebAdminRO** role is used and the operation is attempted under the context of the system user ID that started the web server. If that same user performs a define operation, the **MQWebUser** role is used, and the operation is attempted under the context of the principal.

ALW Changing the certificate presented by the IBM MQ Console to your browser

You can configure the IBM MQ Console to present a CA-signed certificate for authentication purposes. If you configure the IBM MQ Console to present a CA-signed certificate, the browser no longer presents the self-signed certificate warning when the IBM MQ Console is accessed.

About this task

Security for the IBM MQ Console is provided by the mqweb server that runs the IBM MQ Console. To change the certificate that the mqweb server presents to your browser, first add the new certificate to the mqweb server keystore. Then edit the security configuration in the `mqwebuser.xml` file to specify the certificate that the server presents.

The procedure makes the following assumptions:

- You are a privileged user.
- You are using an AIX, Linux, or Windows system.
- That your `mqwebuser.xml` file is based on either the `basic_registry.xml`, `local_os_registry.xml`, or `ldap_registry.xml` sample XML files.

Procedure

1. Optional: Change the default password of the mqweb server keystore `key.jks` by using the **runmqktool** command:

```
runmqktool -storepasswd -keystore MQ_DATA_DIRECTORY/web/installations/installationName/
servers/mqweb/resources/security/key.jks -storepass oldPassword
          -new newPassword
```

   oldPassword
   Specifies the existing `key.jks` password. The default password is `password`.

   newPassword
   Specifies a new `key.jks` password.

2. Create a key pair and certificate request to send to the certificate authority:

   a) Create the key pair by using the **runmqktool** command:

```
runmqktool -genkeypair -keystore MQ_DATA_DIRECTORY/web/installations/installationName/
servers/mqweb/resources/security/key.jks -storepass password -storetype JKS
           -alias label -dname distinguished_name
           -sigalg signature_algorithm
```

password
: Specifies the `key.jks` keystore password.

label
: Specifies the certificate label. For example, `MQWebConsole`.

distinguished_name
: Specifies the X.500 Distinguished Name for the certificate. Enclose the Distinguished Name in double quotation marks.

  For example, `"cn=MQWebConsole,o=myOrg,c=UK"`

signature_algorithm
: Specifies the algorithm to use to sign the certificate. For more information, see Signature algorithms

b) Create the certificate request by using the **`runmqktool`** command:

```
runmqktool -certreq -keystore MQ_DATA_DIRECTORY/web/installations/installationName/
servers/mqweb/resources/security/key.jks -storepass password -alias label
           -file filename
```

password
: Specifies the `key.jks` keystore password.

label
: Specifies the certificate label from substep “2.a” on page 510.

filename
: Specifies the fully qualified file name for the certificate request.

3. Send the certificate request file to a certificate authority (CA).
4. When you have the certificate from the CA, import the certificate and any other certificates in the certificate chain, starting with the root CA certificate, into the `keys.jks` keystore by using the **`runmqktool`** command:

```
runmqktool -importcert -keystore MQ_DATA_DIRECTORY/web/installations/installationName/
servers/mqweb/resources/security/key.jks -storepass password
           -alias label -file filename
```

password
: Specifies the `key.jks` keystore password.

label
: Specifies the certificate label from substep “2.a” on page 510.

filename
: Specifies the fully qualified file name of the certificate to import.

5. Configure the mqweb server to present the CA certificate:

a) Open the `mqwebuser.xml` file.

The `mqwebuser.xml` file can be found on the following path: *`MQ_DATA_PATH`*`/web/installations/`*`installationName`*`/servers/mqweb`

b) Turn off the default security configuration by commenting out the following line:

```
<sslDefault sslRef="mqDefaultSSLConfig"/>
```

If you configured the mqweb server to use client certificate authentication, this line of the xml file is already commented out.

c) Uncomment the section in the `mqwebuser.xml` file that enables custom certificate configuration. The section contains the following text:

```
<keyStore id="defaultKeyStore" location="key.jks" type="JKS" password="password"/>
    <keyStore id="defaultTrustStore" location="trust.jks" type="JKS" password="password"/>
    <ssl id="thisSSLConfig" clientAuthenticationSupported="true"
keyStoreRef="defaultKeyStore"
           trustStoreRef="defaultTrustStore" sslProtocol="TLSv1.2"
```

```
serverKeyAlias="default"/>
    <sslDefault sslRef="thisSSLConfig"/>
```

If you configured the mqweb server to use client certificate authentication, this section of the xml file is already uncommented.

d) Optional: If you changed the password for the `key.jks` keystore in step "1" on page 510, change the value for **password** in the `defaultKeyStore` tags to an encoded version of the password that you set:

i) From the *MQ_INSTALLATION_PATH*`/web/bin` directory, enter the following command:

```
securityUtility encode password
```

ii) Place the output of this command in the **password** field for the `defaultKeyStore`.

e) If you are not using client certificate authentication, comment out the following line:

```
<keyStore id="defaultTrustStore" location="trust.jks" type="JKS" password="password"/>
```

f) Change the value of **serverKeyAlias** from `default` to the value of the CA certificate label.

6. Stop the mqweb server by using the **endmqweb** command.
7. Start the mqweb server by using the **strmqweb** command.

Results

When the web server starts, browse to your IBM MQ Console and refresh. The CA certificate is used and you are taken straight to the login page.

ALW Configuring client certificate authentication with the REST API and IBM MQ Console

You can map client certificates to principals to authenticate IBM MQ Console and REST API users.

Before you begin

- Configure users, groups, and roles to be authorized to use the IBM MQ Console and REST API. For more information, see "Configuring users and roles" on page 498.
- When you use the REST API, you can query the credentials of the current user by using the HTTP GET method on the `login` resource, providing the client certificate to authenticate the request. This request returns information about the username, and the roles that the user is assigned. For more information, see GET `/login`.
- When you map client certificates to principals to authenticate users, the distinguished name of the client certificate is used to match against users in the configured user registry:
  - For a basic registry, the Common Name (CN) is matched against the user. For example, `CN=Fred, O=IBM, C=GB` is matched against a username of `Fred`.
  - For an LDAP registry, by default the full distinguished name is matched against LDAP. You can set up filters and mappings to customize the matching. For more information, see Liberty:LDAP certificate map mode in the WebSphere Liberty documentation.

About this task

When a user authenticates by using a client certificate, the certificate is used in place of a username and password. For the REST API, the client certificate is provided with each REST request to authenticate the user. For the IBM MQ Console, when a user logs in with a certificate, the user cannot then be logged out.

ALW On AIX, Linux, or Windows systems, the procedure assumes the following information:

- That your `mqwebuser.xml` file is based on either the `basic_registry.xml`, `local_os_registry.xml`, or `ldap_registry.xml` sample XML files.
- That you are a privileged user.

z/OS To configure client certificate authentication with a RACF key ring on z/OS systems, follow the procedure in "Configuring TLS for the REST API and IBM MQ Console on z/OS" on page 525.

Note: The following procedure outlines the steps necessary to use client certificates with the IBM MQ Console and REST API. For developer convenience, the steps detail how to create and use self-signed certificates. However, for production, use certificates that are obtained from a certificate authority.

Procedure

1. Create a certificate by using the **`runmqktool`** command:

   ```
   runmqktool -genkeypair -keystore filename -storepass password -storetype PKCS12
              -alias label -dname distinguished_name
              -sigalg signature_algorithm
   ```

   filename
   Specifies the keystore name, for example `user.p12`. If the keystore does not exist, it is created when the command runs.

   password
   Specifies the keystore password.

   label
   Specifies the certificate label. For example, `user1`.

   distinguished_name
   Specifies the X.500 Distinguished Name for the certificate. Enclose the Distinguished Name in double quotation marks.

   If you are using a basic user registry, enter the name of a user from your user registry in the Common Name (CN) part of the Distinguished Name. For example, for a user `mqadmin`, use the Distinguished Name `"CN=mqadmin"`.

   If you are using a local OS registry, enter the name of a local OS user ID in the Common Name (CN) part of the Distinguished Name. For example, for a user `mqadmin`, use the Distinguished Name `"CN=mqadmin"`.

   If you are using an LDAP user registry, enter a Distinguished Name that matches the Distinguished Name in the LDAP registry.

   signature_algorithm
   Specifies the algorithm to use to sign the certificate. For more information, see Signature algorithms

2. Optional: Obtain a certificate from a certificate authority (CA). Alternatively, to use a self-signed certificate, continue to step "3" on page 514.

   a) To obtain a certificate from a certificate authority, create a certificate request by using the **`runmqktool`** command:

   ```
   runmqktool -certreq -keystore filename -storepass password -alias label
              -file filename
   ```

   filename
   Specifies the keystore name from step "1" on page 513.

   password
   Specifies the keystore password.

   label
   Specifies the certificate label from step "1" on page 513.

   filename
   Specifies the fully qualified file name for the certificate request.

b) Send the certificate request file to a certificate authority (CA).

c) When you have the certificate from the CA, import the certificate into your keystore by using the **runmqktool** command:

```
runmqktool -importcert -keystore filename -storepass password
           -alias label -file filename
```

filename
Specifies the keystore name from step "1" on page 513.

password
Specifies the keystore password.

label
Specifies the certificate label from step "1" on page 513.

filename
Specifies the fully qualified file name of the CA certificate.

3. Extract the public part of the certificate by using the **runmqktool** command:

```
runmqktool -exportcert -keystore filename -storepass password
           -alias label -file filename -rfc
```

filename
Specifies the keystore name from step "1" on page 513.

password
Specifies the keystore password.

label
Specifies the certificate label from step "1" on page 513.

filename
Specifies the fully qualified file name for the extracted certificate.

4. Import the public part of the certificate into the mqweb server trust keystore as a signer certificate so that the server can validate the client certificate by using the **runmqktool** command:

```
runmqktool -importcert -keystore  MQ_DATA_DIRECTORY/web/installations/installationName/
servers/mqweb/resources/security/trust.jks -storepass password
           -alias label -file filename
```

password
Specifies the `trust.jks` keystore password. You can specify either a password for an existing `trust.jks` keystore, or a new password for a new `trust.jks` keystore.

label
Specifies the certificate label from step "1" on page 513.

filename
Specifies the fully qualified file name of the extracted certificate.

5. Configure the mqweb server to use client certificate authentication:

a) Open the `mqwebuser.xml` file.

The `mqwebuser.xml` file can be found on the following path: *MQ_DATA_PATH*`/web/installations/`*installationName*`/servers/mqweb`

b) Turn off the default security configuration by commenting out the following line:

```
<sslDefault sslRef="mqDefaultSSLConfig"/>
```

If you configured the mqweb server to present a CA certificate to the browser, this line is already commented out.

c) Uncomment the section in the `mqwebuser.xml` file that enables client certificate authentication. The section contains the following text:

```
<keyStore id="defaultKeyStore" location="key.jks" type="JKS" password="password"/>
    <keyStore id="defaultTrustStore" location="trust.jks" type="JKS" password="password"/>
    <ssl id="thisSSLConfig" clientAuthenticationSupported="true"
keyStoreRef="defaultKeyStore"
           trustStoreRef="defaultTrustStore" sslProtocol="TLSv1.2"
serverKeyAlias="default"/>
    <sslDefault sslRef="thisSSLConfig"/>
```

If you configured the mqweb server to present a CA certificate to the browser, this section is already uncommented. However, you might need to uncomment the **defaultTrustStore** line.

d) Change the value for **password** for the `defaultTrustStore` to match the password for the `trust.jks` keystore:

i) From the *MQ_INSTALLATION_PATH*`/web/bin` directory, enter the following command:

```
securityUtility encode password
```

ii) Place the output of this command in the **password** field for the `defaultTrustStore`.

6. Stop the mqweb server by using the **endmqweb** command.
7. Start the mqweb server by using the **`strmqweb`** command.
8. Use the client certificate to authenticate:

   - To use the client certificate with the IBM MQ Console, install the client certificate into the web browser that is used to access the IBM MQ Console.
   - To use the client certificate with the REST API, provide the client certificate with each REST request. When you use HTTP POST, PATCH, or DELETE methods, you must provide extra authentication with the client certificate to prevent cross-site request forgery attacks. That is, the extra authentication is used to confirm that the credentials that are being used to authenticate the request are being used by the owner of the credentials.

     This extra authentication is provided by the `ibm-mq-rest-csrf-token` HTTP header. Set the value of the `ibm-mq-csrf-token` header to anything, including blank, then submit the request.

Example

Important: In the example, not all cURL implementations support self-signed certificates, so you must use a cURL implementation that does.

The following cURL example shows how to create a new queue `Q1`, on a queue manager `QM1`, with client certificate authentication. The exact configuration of this cURL command depends on the libraries that cURL was built with. The example is based on a Windows system with cURL built against OpenSSL.

- Use the HTTP POST method with the queue resource, authenticating with the client certificate and including the `ibm-mq-rest-csrf-token` HTTP header with an arbitrary value. This value can be anything, including blank. The `--cert-type` flag specifies that the certificate is a PKCS#12 certificate. The `--cert` flag specifies the location of the certificate, followed by a colon, and then the password for the certificate:

```
curl -k https://localhost:9443/ibmmq/rest/v1/admin/qmgr/QM1/queue -X POST -
-cert-type P12 --cert c:\user.p12:password
-H "ibm-mq-rest-csrf-token: value"
-H "Content-Type: application/json" --data "{\"name\":\"Q1\"}"
```

Using HTTP basic authentication with the REST API

Users of the REST API can authenticate by providing their user ID and password within an HTTP header. To use this method of authentication with HTTP methods, such as POST, PATCH, and DELETE, the `ibm-mq-rest-csrf-token` HTTP header must also be provided, as well as a user ID and password.

Before you begin

- Configure users, groups, and roles to be authorized to use the REST API. For more information, see "Configuring users and roles" on page 498.
- Ensure that HTTP basic authentication is enabled. Check that the following XML is present, and is not commented out, in the `mqwebuser.xml` file. This XML must be within the `<featureManager>` tags:

  ```
  <feature>basicAuthenticationMQ-1.0</feature>
  ```

  z/OS On z/OS, you must be a user that has write access to `mqwebuser.xml` to edit this file.

  Multi On all other operating systems, you must be a privileged user to edit the `mqwebuser.xml` file.
- Ensure that you are using a secure connection when you send REST requests. As the user name and password combination are encoded, but not encrypted, you must use a secure connection (HTTPS) when you use HTTP basic authentication with the REST API.
- You can query the credentials of the current user by using the HTTP GET method on the `login` resource, providing the basic authentication information to authenticate the request. This request returns information about the user name, and the roles that the user is assigned. For more information, see GET `/login`.

Procedure

1. Concatenate the user name with a colon, and the password. Note that the user name is case-sensitive.

   For example, a user name of admin, and a password of admin becomes the following string:

   ```
   admin:admin
   ```
2. Encode this user name and password string in base64 encoding.
3. Include this encoded user name and password in an HTTP `Authorization: Basic` header.

   For example, with an encoded user name of admin, and a password of admin, the following header is created:

   ```
   Authorization: Basic YWRtaW46YWRtaW4=
   ```
4. When you use HTTP POST, PATCH, or DELETE methods, you must provide extra authentication, as well as a user name and password.

   This extra authentication is provided by the `ibm-mq-rest-csrf-token` HTTP header. The `ibm-mq-rest-csrf-token` HTTP header must be present in the request, but its value can be anything, including blank.
5. Submit your REST request to IBM MQ with the appropriate headers.

Example

The following example shows how to create a new queue Q1, on queue manager QM1, with basic authentication, on Windows systems. The example uses cURL:

- Use the HTTP POST method with the queue resource, authenticating with basic authentication and including the `ibm-mq-rest-csrf-token` HTTP header with an arbitrary value. This value can be anything, including blank:

  ```
  curl -k https://localhost:9443/ibmmq/rest/v1/admin/qmgr/QM1/queue -X POST
  -u mqadmin:mqadmin
  -H "ibm-mq-rest-csrf-token: value"
  -H "Content-Type: application/json" --data "{\"name\":\"Q1\"}"
  ```

Using token-based authentication with the REST API

Users of the REST API can authenticate by providing a user ID and password to the REST API `login` resource with the HTTP POST method. An LTPA token is generated that enables the user to authenticate future requests. This LTPA token has the prefix `LtpaToken2`. The user can log out by using the HTTP DELETE method, and can query the log in information of the current user with the HTTP GET method.

Before you begin

- Configure users, groups, and roles to be authorized to use the REST API. For more information, see "Configuring users and roles" on page 498.
- By default, the name of the cookie that includes the LTPA token starts with `LtpaToken2`, and includes a suffix that can change when the mqweb server is restarted. This randomized cookie name allows more than one mqweb server to run on the same system. However, if you want the cookie name to remain a consistent value, you can specify the name that the cookie has by using the **setmqweb** command. For more information, see Configuring the LTPA token.
- By default, the LTPA token cookie expires after 120 minutes. You can configure the expiry time of the LTPA token cookie by using the **setmqweb** command. For more information, see Configuring the LTPA token.
- Ensure that you are using a secure connection when you send REST requests. When you use the HTTP POST method on the `login` resource, the user name and password combination that is sent with the request are not encrypted. Therefore, you must use a secure connection (HTTPS) when you use token based authentication with the REST API. By default, you cannot use HTTP with LTPA token authentication. You can enable the LTPA token to be used by insecure HTTP connections by setting **secureLTPA** to `False`. For more information, see Configuring the LTPA token.
- You can query the credentials of the current user by using the HTTP GET method on the `login` resource, providing the LTPA token to authenticate the request. This request returns information about the user name, and the roles that the user is assigned. For more information, see GET `/login`.

Procedure

1. Log in a user:
   a) Use the HTTP POST method on the `login` resource:

      ```
      https://host:port/ibmmq/rest/v1/login
      ```

      Include the user name and password in the body of the JSON request, in the following format:

      ```
      {
          "username" : name,
          "password" : password
      }
      ```

   b) Store the LTPA token that is returned from the request in the local cookie store. By default, this LTPA token has a prefix of `LtpaToken2`.
2. Authenticate REST requests with the stored LTPA token as a cookie with every request.

   For requests that use the HTTP PUT, PATCH, or DELETE methods, include an `ibm-mq-rest-csrf-token` header. The value of this header can be anything, including blank.

3. Log out a user:
    a) Use the HTTP DELETE method on the `login` resource:

    ```
    https://host:9443/ibmmq/rest/v1/login
    ```

    You must provide the LTPA token as a cookie to authenticate the request, and include an `ibm-mq-rest-csrf-token` header. The value of this header can be anything, including blank

    b) Process the instruction to delete the LTPA token from the local cookie store.

    Note: If the instruction is not processed, and the LTPA token remains in the local cookie store, then the LTPA token can be used to authenticate future REST requests. That is, when the user attempts to authenticate with the LTPA token after the session is ended, a new session is created that uses the existing token.

Example

The following cURL example shows how to create a new queue Q1, on queue manager QM1, with token-based authentication, on Windows systems:

- Log in and add the LTPA token with the prefix `LtpaToken2`, to the local cookie store. The user name and password information are included in the JSON body. The `-c` flag specifies the location of the file to store the token in:

```
curl -k https://localhost:9443/ibmmq/rest/v1/login -X POST
-H "Content-Type: application/json" --data
"{\"username\":\"mqadmin\",\"password\":\"mqadmin\"}"
-c c:\cookiejar.txt
```

- Create a queue. Use the HTTP POST method with the queue resource, authenticating with the LTPA token. The LTPA token with the prefix `LtpaToken2` is retrieved from the `cookiejar.txt` file by using the `-b` flag. CSRF protection is provided by the presence of the `ibm-mq-rest-csrf-token` HTTP header:

```
curl -k https://localhost:9443/ibmmq/rest/v1/admin/qmgr/QM1/queue -X POST -b
c:\cookiejar.txt -H "ibm-mq-rest-csrf-token: value" -H "Content-Type: application/json"
--data "{\"name\":\"Q1\"}"
```

- Log out and delete the LTPA token from the local cookie store. The LTPA token is retrieved from the `cookiejar.txt` file by using the `-b` flag. CSRF protection is provided by the presence of the `ibm-mq-rest-csrf-token` HTTP header. The location of the `cookiejar.txt` file is specified by the `-c` flag so that the LTPA token is deleted from the file:

```
curl -k https://localhost:9443/ibmmq/rest/v1/admin/qmgr/QM1/queue -X DELETE
-H "ibm-mq-rest-csrf-token: value" -b c:\cookiejar.txt
-c c:\cookiejar.txt
```

Related reference

POST /login

GET /login

DELETE /login

Embedding the IBM MQ Console in an IFrame

The HTML <iframe> element can be used to embed one web page into another using an Inline Frame (IFrame). For security reasons the IBM MQ Console can not be embedded into an IFrame by default.

However, you can enable an IFrame by using the **mqConsoleFrameAncestors** configuration property on the mqweb server.

About this task

The mqweb server maintains an allowlist of origins of web pages that can embed the IBM MQ Console using an IFrame. An origin is a combination of a URL scheme, domain and port, for example, `https://example.com:1234`.

You can use the **mqConsoleFrameAncestors** configuration property on the mqweb server to specify the entries in the list.

By default, **mqConsoleFrameAncestors** is blank, which means that the IBM MQ Console can not be embedded in an IFrame.

Procedure

Specify a list of origins of web pages, that can embed the IBM MQ Console in an IFrame, by entering the following command:

```
setmqweb properties -k mqConsoleFrameAncestors -v allowedOrigins
```

where *allowedOrigins* is a comma separated list of origins. Each origin should consist of:

- A hostname or IP address
- An optional URL scheme
- An optional port number

Note that the host name can start with the wildcard character (*) and the port number can also use the wild card character (*).
Example origins are:

```
https://example.com:1234
```

which allows any web page served from `https://example.com:1234` to embed the IBM MQ Console in an IFrame.

```
https://*.example.com:*
```

which allows any HTTPS web page with a hostname ending with `example.com`, and using any port, to embed the IBM MQ Console in an IFrame.

Example

The following example allows the IBM MQ Console to be embedded in an IFrame from web pages served from either `https://site2.example.com:1234` or `https://site2.example.com:1235`:

```
setmqweb properties -k mqConsoleFrameAncestors -v
https://site2.example.com:1234,https://site2.example.com:1235
```

Configuring CORS for the REST API

By default, a web browser does not allow scripts, such as JavaScript, to invoke the REST API when the script is not from the same origin as the REST API. That is, cross-origin requests are not enabled. You can configure Cross Origin Resource Sharing (CORS) to allow cross-origin requests from specified origins.

About this task

You can access the REST API through a web browser, for example through a script. As these requests are from a different origin to the REST API, the web browser refuses the request because it is a cross-origin request. The origin is different if the domain, port, or scheme is not the same.

For example, if you have a script that is hosted at `http://localhost:1999/` you make a cross-origin request if you issue an HTTP GET on a website that is hosted at `https://localhost:9443/`. This request is a cross-origin request because the port numbers and scheme (HTTP) are different.

You can enable cross-origin requests by configuring CORS and specifying the origins that are allowed to access the REST API.

For more information about CORS, see https://www.w3.org/TR/cors/ and https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS.

Procedure

1. View the current configuration by entering the following command:

   ```
   dspmqweb properties -a
   ```

   The `mqRestCorsAllowedOrigins` entry specifies the allowed origins. The `mqRestCorsMaxAgeInSeconds` entry specifies the time, in seconds, that the web browser can cache the results of any CORS pre-flight checks.
2. Specify the origins that are allowed to access the REST API by entering the following command:

   ```
   setmqweb properties -k mqRestCorsAllowedOrigins -v allowedOrigins
   ```

   where *allowedOrigins* specifies the origin that you want to allow cross-origin requests from. You can use an asterisk surrounded by double quotation marks, "*", to allow all cross-origin requests. You can enter more than one origin in a comma-separated list, surrounded by double quotation marks. To allow no cross-origin requests, enter empty quotation marks as the value for *allowedOrigins*.
3. Specify the time, in seconds, that you want to allow a web browser to cache the results of any CORS pre-flight checks by entering the following command:

   ```
   setmqweb properties -k mqRestCorsMaxAgeInSeconds -v time
   ```

Example

The following example shows cross-origin requests enabled for `http://localhost:9883`, `https://localhost:1999`, and `https://localhost:9663`. The maximum age of cached results of any CORS pre-flight checks is set to 90 seconds:

```
setmqweb properties -k mqRestCorsAllowedOrigins -v "http://localhost:9883,https://
localhost:1999,https://localhost:9663"
setmqweb properties -k mqRestCorsMaxAgeInSeconds -v 90
```

Configuring host header validation for the IBM MQ Console and REST API

You can configure the mqweb server to restrict access to the IBM MQ Console and REST API such that only requests that are sent with a host header that matches a specified allowlist are processed. An error is returned if a host header value that is not on the allowlist is used.

About this task

The mqweb server uses virtual hosts to define the allowlist of acceptable host headers. For more information about virtual hosts, see the WebSphere Liberty documentation: https://www.ibm.com/docs/SSEQTP_liberty/com.ibm.websphere.wlp.doc/ae/cwlp_virtual_hosts.html

To complete this task, you must be a user with sufficient privileges to edit the `mqwebuser.xml` file:

- z/OS On z/OS, you must have write access to the `mqwebuser.xml` file.
- Multi On all other operating systems, you must be a privileged user.
- Linux V 9.4.0 If the mqweb server is part of a stand-alone IBM MQ Web Server installation, you must have write access to the `mqwebuser.xml` file in the IBM MQ Web Server data directory.

Procedure

1. Open the `mqwebuser.xml` file. This file is in one of the following locations:
   - In an IBM MQ installation:
     - Linux AIX On AIX or Linux: `/var/mqm/web/installations/`*`installationName`*`/servers/mqweb`
     - Windows On Windows: *`MQ_DATA_PATH`*`\web\installations\`*`installationName`*`\servers\mqweb`, where *MQ_DATA_PATH* is the IBM MQ data path. This path is the data path that is selected during the installation of IBM MQ. By default, this path is `C:\ProgramData\IBM\MQ`.
     - z/OS On z/OS: *`WLP_user_directory`*`/servers/mqweb`

       Where *WLP_user_directory* is the directory that was specified when the **`crtmqweb`** command ran to create the mqweb server definition.
   - Linux V 9.4.0 In a stand-alone IBM MQ Web Server installation: *`MQ_OVERRIDE_DATA_PATH`*`/web/installations/MQWEBINST/servers/mqweb`

     where *MQ_OVERRIDE_DATA_PATH* is the IBM MQ Web Server data directory that the **`MQ_OVERRIDE_DATA_PATH`** environment variable points to.
2. Add or uncomment the following code in the `mqwebuser.xml` file:

```
<virtualHost allowFromEndpointRef="defaultHttpEndpoint" id="default_host">
        <hostAlias>localhost:9080</hostAlias>
</virtualHost>
```

3. Edit the **`<hostAlias>`** field, inserting the hostname and port combination that you want to allow.

   This combination might be the hostname and port name that you used in the configuration of the mqweb server. For example, if you use the default configuration of `localhost:9443`, you might want to use `localhost:9443` in the **`<hostAlias>`** field.

   If necessary, you can add multiple **`<hostAlias>`** fields within the **`<virtualHost>`** tags to allow more hostname and port combinations. For example, to allow host headers that use an HTTP port as well as host headers that use the HTTPS port.

Auditing

Audit records of operations that are performed in the IBM MQ Console and REST API can be produced by enabling queue manager command and configuration events, and on AIX, Linux, and Windows significant state changes are recorded in the log files of the mqweb server.

Significant state changes

ALW

On AIX, Linux, and Windows, the IBM MQ Console records significant state changes as messages in the logs of the mqweb server. Each message indicates the authenticated principal name that requested the operation.

Significant state changes, such as when queue managers are created, started, ended, or deleted, are logged in the mqweb server `messages.log` and `console.log` files at the [AUDIT] logging level. Each log entry indicates the authenticated principal name that requested the operation.

The `messages.log` and `console.log` files can be found in the following location:

- In an IBM MQ installation:
  - Linux AIX On AIX or Linux: `/var/mqm/web/installations/`*`installationName`*`/servers/mqweb/logs`
  - Windows On Windows: *`MQ_DATA_PATH`*`\web\installations\`*`installationName`*`\servers\mqweb\logs`, where *MQ_DATA_PATH* is the IBM MQ data path. This path is the data path that is selected during the installation of IBM MQ. By default, this path is `C:\ProgramData\IBM\MQ`.
- Linux V 9.4.0 In a stand-alone IBM MQ Web Server installation: *`MQ_OVERRIDE_DATA_PATH`*`/web/installations/MQWEBINST/servers/mqweb/logs`

  where *MQ_OVERRIDE_DATA_PATH* is the IBM MQ Web Server data directory that the **`MQ_OVERRIDE_DATA_PATH`** environment variable points to.

For more information about configuring the mqweb server logging levels, see Configuring logging.

Command and configuration events

You can optionally enable command and configuration events on the queue manager to provide information about most IBM MQ Console and REST API activity. For example, the creation of channels, and the inquiry of queues generate command and configuration events. For more information about enabling command and configuration events, see Controlling configuration, command and logger events.

For these command and configuration event messages, the **`MQIACF_EVENT_ORIGIN`** field is set to `MQEVO_REST` and the **`MQCACF_EVENT_APPL_IDENTITY`** field reports the first 32 characters of the authenticated principal name. If a user has the `MQWebAdmin` or `MQWebAdminRO` role, the **`MQCACF_EVENT_USER_ID`** field reports the mqweb server user ID, not the username of the principal that issued the command. However, if the user has the `MQWebUser` role, the **`MQCACF_EVENT_USER_ID`** reports the username of the principal that issued the command.

Related concepts

"Auditing" on page 466

You can check for security intrusions, or attempted intrusions, by using event messages. You can also check the security of your system by using the IBM MQ Explorer.

z/OS Security considerations for the IBM MQ Console and REST API on z/OS

The IBM MQ Console and REST API have security features controlling whether a user can issue, display, or alter commands. The commands are then passed to the queue manager, and the queue manager security is then used to control if the user is allowed to issue the command to that specific queue manager.

Procedure

1. Ensure that the mqweb server started task user ID has appropriate authorities to issue certain PCF commands and access certain queues. For more information, see "Authority required by the mqweb server started task user ID" on page 523.
2. Ensure that any users that are granted the `MQWebUser` role have appropriate authorities.

   IBM MQ Console and REST API users that are assigned to the `MQWebUser` role operate under the security context of the principal. These user IDs can only perform operations that the user ID is granted to perform on the queue manager, and need to be granted access to the same system queues as the mqweb server address space.

   The mqweb server started task user ID must be granted alternate user access to all users assigned to the `MQWebUser` role.

   For more information about granting appropriate authorities for users with the `MQWebUser` role, see "Access to IBM MQ resources required to use the IBM MQ Console or REST API " on page 524.
3. Optional: Configure TLS for the IBM MQ Console and REST API. For more information, see "Configuring TLS for the REST API and IBM MQ Console on z/OS" on page 525.

z/OS Authority required by the mqweb server started task user ID

On z/OS, the mqweb server started task user ID requires certain authorities to issue PCF commands and access system resources.

The mqweb server started task user ID needs:

- A z/OS UNIX user identifier (UID) to be able to use z/OS UNIX System Services.
- Access to the `hlq.SCSQAUTH` and `hlq.SCSQANL*` data sets in the IBM MQ installation.
- Read access to the IBM MQ installation files in z/OS UNIX System Services.
- Read and write access to the Liberty user directory created by the **`crtmqweb`** script.
- Authority to connect to the queue manager. Grant the mqweb server started task user ID *READ* access to the `hlq.BATCH` profile in the MQCONN class.
- Authority to issue IBM MQ commands and access certain queues. These details are described in "IBM MQ Console - required command security profiles" on page 226, "System queue security" on page 204, and "Profiles for context security" on page 214.
- Authority to subscribe to the `SYSTEM.FTE` topic, in order to use the REST API for MFT. Grant the mqweb server started task user ID *ALTER* access to the `hlq.SUBSCRIBE.SYSTEM.FTE` profile in the MXTOPIC class.
- If you are are configuring a SAF registry, access to various security profiles. See "Configuring a SAF registry for the IBM MQ Console and REST API" on page 506 for more information.

Connection authentication

If your queue manager has been configured to require that all batch applications provide a valid user ID and password, by setting CHKLOCL(REQUIRED), you must give the mqweb server started task user ID *UPDATE* access to the `hlq.BATCH` profile in the MQCONN class.

This authority causes connection authentication to operate in CHKLOCL(OPTIONAL) mode for the mqweb server started task user ID.

If you have not configured the queue manager to require that all batch applications provide a valid user ID and password, it is sufficient to give the user ID that starts the mqweb server task *READ* access to the `hlq.BATCH` profile in the MQCONN class.

For more information about CHCKLOCL, see "Using CHCKLOCL on locally bound applications" on page 195.

z/OS Access to IBM MQ resources required to use the IBM MQ Console or REST API

Operations performed in the IBM MQ Console, or REST API, by a user in the `MQWebUser` role take place under the security context of the user.

About this task

See "Roles on the IBM MQ Console and REST API" on page 508 for more information on the roles in the IBM MQ Console and REST API.

Use the following procedure to grant a user, in the `MQWebUser` role, access to the queue manager resources required to use the IBM MQ Console or REST API.

Procedure

1. Grant the `mqweb server started task` user ID alternate user access to each user ID in the `MQWebUser` role.

   Do this on every queue manager that users will administer through the IBM MQ Console or REST API.

   You can use the following sample RACF commands to grant the `mqweb server started task` user ID alternate user access to a user in the `MQWebUser` role:

   ```
   RDEFINE MQADMIN hlq.ALTERNATE.USER.userId UACC(NONE)
   PERMIT hlq.ALTERNATE.USER.userId CLASS(MQADMIN) ACCESS(UPDATE) ID(mqwebUserId)
   SETROPTS RACLIST(MQADMIN) REFRESH
   ```

   where:

   `hlq`
   Is the profile prefix, that can be either the queue manager name, or queue sharing group name

   `userId`
   Is the user in the `MQWebUser` role

   `mqwebUserId`
   Is the `mqweb server started task` user ID

   Note: If you are using mixed-case security, use the MXADMIN class rather than the MQADMIN class.

2. Grant each user in the `MQWebUser` role access to system queues that are necessary to use the IBM MQ Console and REST API.

   To do this, for both the SYSTEM.ADMIN.COMMAND.QUEUE and SYSTEM.REST.REPLY.QUEUE, give each user UPDATE access to the MQQUEUE or MXQUEUE classes, depending on whether mixed-case security is in use.

   You need to do this on every queue manager that the user will administer through the REST API, including remote queue managers administered through the administrative REST API gateway.

3. To allow a user in the `MQWebUser` role to administer remote queue managers, grant the user UPDATE access to the profile in the MQQUEUE or MXQUEUE class, protecting the transmission queue used to send commands to the remote queue manager. Note that you need to give the user UPDATE access on the gateway queue manager.

On the remote queue manager, grant access for the same user, to put to the transmission queue used to send command response messages back to the gateway queue manager.

4. Grant the users in the `MQWebUser` role access to any other resources required to perform the operations supported by the IBM MQ Console and REST API.

   The access needed to:

   - Perform operations in the REST API, is described in the *Security requirements* sections of the individual REST API resources
   - Issue commands by the IBM MQ Console is described in "IBM MQ Console - required command security profiles" on page 226

z/OS Configuring TLS for the REST API and IBM MQ Console on z/OS

On z/OS, you can configure the mqweb server to use a RACF key ring to store certificates for secure connections with TLS, and client certificate authentication.

Before you begin

You must be a user that has write access to the `mqwebuser.xml` file, and authority to work with SAF key rings, to complete this procedure.

About this task

The default mqweb server configuration uses Java keystores for the server and trusted certificates. On z/OS, you can configure the mqweb server to use a RACF key ring, instead of the Java keystores. The server can also be configured to allow users to authenticate using a client certificate.

See Liberty: Keystores for information on using RACF key rings in Liberty.

Follow this procedure to configure the mqweb server to use a RACF key ring, and optionally configure client certificate authentication. This procedure describes the steps necessary to create and use certificates signed with your own certificate authority (CA) certificates. For production, you might prefer to use certificates obtained from an external certificate authority.

Procedure

1. Create a certificate authority (CA) certificate, which will be used to sign the server certificate. For example, enter the following RACF command:

```
RACDCERT GENCERT -
  CERTAUTH -
  SUBJECTSDN(CN('mqweb Certification Authority') -
    O('IBM') -
    OU('MQ')) -
  SIZE(2048) -
  WITHLABEL('mqwebCertauth')
```

2. Create a server certificate, signed with the CA certificate created in step 1, by entering the following command:

```
RACDCERT ID(mqwebUserId) GENCERT -
  SUBJECTSDN(CN('hostname') -
   O('IBM') -
   OU('MQ')) -
   SIZE(2048) -
   SIGNWITH (CERTAUTH LABEL('mqwebCertauth')) -
  WITHLABEL('mqwebServerCert')
```

   where *mqwebUserId* is the mqweb server started task user ID, and *hostname* is the host name of the mqweb server.

3. Connect the CA certificate and server certificate to a SAF key ring by entering the following commands:

```
RACDCERT ID(mqwebUserId) CONNECT(RING(keyring) LABEL('mqwebCertauth') CERTAUTH)
RACDCERT ID(mqwebUserId) CONNECT(RING(keyring) LABEL('mqwebServerCert'))
```

where *mqwebUserId* is the mqweb server started task user ID, and *keyring* is the name of the key ring you want to use.

4. Export the CA certificate to a CER file by entering the following command:

```
RACDCERT CERTAUTH EXPORT(LABEL('mqwebCertauth')) -
        DSN('hlq.CERT.MQWEBCA') -
        FORMAT(CERTDER) -
        PASSWORD('password')
```

5. FTP the exported CA certificate in binary to your workstation, and import it into your browser as a certificate authority certificate.
6. Optional: If you want to configure client certificate authentication, create and export a client certificate.
   a) Create a certificate authority (CA) certificate, which will be used to sign the client certificate. For example, enter the following RACF command:

   ```
   RACDCERT GENCERT -
     CERTAUTH -
     SUBJECTSDN(CN('mqweb User CA') -
      O('IBM') -
      OU('MQ')) -
      SIZE(2048) -
     WITHLABEL('mqwebUserCertauth')
   ```

   b) Connect the CA certificate to a SAF key ring by entering the following command:

   ```
   RACDCERT ID(mqwebUserId) CONNECT(RING(keyring) LABEL('mqwebUserCertauth') CERTAUTH)
   ```

   where *mqwebUserId* is the mqweb server started task user ID, and *keyring* is the name of the key ring you want to use.

   c) Create a client certificate, signed with the CA certificate. For example, enter the following command:

   ```
   RACDCERT ID(clientUserId) GENCERT -
     SUBJECTSDN(CN('clientUserId') -
        O('IBM') -
        OU('MQ')) -
      SIZE(2048) -
      SIGNWITH (CERTAUTH LABEL('mqwebUserCertauth')) -
     WITHLABEL('userCertLabel')
   ```

   where *clientUserId* is the user name.

   The method used to map a certificate to a principal depends on the type of user registry configured:

   - If you are using a basic registry, the Common Name field in the certificate is matched against the user in the registry.
   - If you are using a SAF registry, and the certificate is in the RACF database, the certificate owner, specified with the **ID** parameter when creating the certificate, is used.
   - If you are using an LDAP registry, the full distinguished name in the certificate is matched against the LDAP registry.

   d) Export the client certificate to a PKCS #12 file by entering the following command:

   ```
   RACDCERT ID(mqwebUserId) EXPORT(LABEL('userCertLabel')) -
           PASSWORD('password') DSN('hlq.USER.CERT')
   ```

   e) FTP the exported certificate in binary to your workstation. To use the client certificate with the IBM MQ Console, import it into the web browser used to access the IBM MQ Console as a personal certificate.

7. Edit the file *WLP_user_directory*/servers/mqweb/mqwebuser.xml, where *WLP_user_directory* is the directory that was specified when the **crtmqweb** script ran to create the mqweb server definition.

   Make the following changes to configure the mqweb server to use a RACF key ring:

   a) Remove, or comment out, the following line:

   ```
   <sslDefault sslRef="mqDefaultSSLConfig"/>
   ```

   b) Add the following statements:

   ```
   <keyStore id="defaultKeyStore" filebased="false"
             location="safkeyring://mqwebUserId/keyring"
             password="password" readOnly="true" type="JCERACFKS" />
   <ssl id="thisSSLConfig" keyStoreRef="defaultKeyStore" sslProtocol="TLSv1.2"
             serverKeyAlias="mqwebServerCert" clientAuthenticationSupported="true" />
   <sslDefault sslRef="thisSSLConfig"/>
   ```

   where:

   - *mqwebUserId* is the mqweb server started task user ID.
   - *keyring* is the name of the RACF key ring.
   - *mqwebServerCert* is the label of the mqweb server certificate.

   Notes: The value of **keyStore password** is ignored.
8. Restart the mqweb server by stopping and restarting the mqweb server started task.
9. Optional: Use the client certificate to authenticate:

   - To use the client certificate with the IBM MQ Console, enter the URL for the IBM MQ Console in the web browser where you installed the client certificate.
   - To use the client certificate with the REST API, provide the client certificate with each REST request.

   Notes:

   a. If you are using only certificates to authenticate to the IBM MQ Console, the browser might display a list of certificates for you to select from.
   b. If you want to use a different certificate you might need to close and restart your browser.
   c. If you are using client certificates that are not in the RACF database, you can use RACF certificate name filtering, to map certificate attributes to a user ID. For example:

   ```
   RACDCERT ID(DEPT3USR) MAP SDNFILTER(OU=DEPT1.C=US)
   ```

   maps certificates with a subject distinguished name containing OU=DEPT1 and C=US to user ID DEPT3USR.

Results

You have set up a TLS interface for the IBM MQ Console and REST API.

ALW Managing keys and certificates on AIX, Linux, and Windows

On AIX, Linux, and Windows, use the **runmqakm** and V 9.4.0 V 9.4.0 **runmqktool** commands to manage keys, certificates, and certificate requests.

About this task

The **runmqakm** command provides functions that are similar to those of **gskitcapicmd**.
V 9.4.0 V 9.4.0 The **runmqktool** command provides functions similar to those of the Java

keytool certificate management utility. Before using the **runmqakm** or **runmqktool** commands, ensure that the systems environment variables are correctly configured by running the **setmqenv** command.

The **runmqktool** command requires the IBM MQ JRE component to be installed. If this component is not installed you can use the **runmqakm** command instead.

If you need to manage TLS certificates in a way that is FIPS compliant, use the **runmqakm** command. This is because the **runmqakm** command supports stronger encryption.

Procedure

- Use the **runmqakm** and **runmqktool** commands to complete the following actions:
  - Create a CMS and PKCS #12 key repository that IBM MQ supports.
  - Create certificate requests.
  - Export certificates.
  - Import personal certificates and CA certificates.
  - Manage self-signed certificates.
  - Create, extract, and add secret keys.

Related information
Keytool

ALW runmqakm and runmqktool commands on AIX, Linux, and Windows

On AIX, Linux, and Windows systems, use the **runmqakm** (GSKCapiCmd) or **runmqktool** (keytool) commands to manage keys and certificates.

Note: V 9.4.0 V 9.4.0

From IBM MQ 9.4.0, the **runmqckm** and **strmqikm** commands are removed. The **runmqktool** command can be used instead of the **runmqckm** command to manage PKCS #12 and JKS key repositories. There is no replacement for the **strmqikm** GUI.

The **runmqckm** and **runmqktool** commands have the following important differences:

- The **runmqktool** command does not support stash files to store key repository passwords. The password to access a key repository must always be provided to the **runmqktool** command when it is run, either as a parameter to the command, or in response to a prompt issued by the command.
- The **runmqktool** command does not support CMS key repositories. Therefore, to export a certificate from a JKS to a CMS key repository, you must complete following steps:
  1. Use the **runmqktool -importkeystore** command to copy the certificate from the JKS key repository to an intermediate PKCS #12 key repository. For more information about exporting a certificate, see "Exporting a personal certificate from a key repository on AIX, Linux, and Windows" on page 538.
  2. Use the **runmqakm -cert -import** command to import the certificate from the intermediate PKCS #12 key repository to the CMS key repository. For more information about importing a certificate, see "Importing a personal certificate into a key repository on AIX, Linux, and Windows" on page 539.

The following IBM MQ commands can be used to manage keys and certificates:

runmqakm

- Provides functions that are similar to those of **gskitcapicmd**.
- Supports CMS and PKCS #12 key repositories.
- Supports the creation of a stash file to store the encrypted key repository password.

- Certified as FIPS 140-2 compliant, and can be configured to operate in a FIPS-compliant manner with the **-fips** parameter.

V 9.4.0 V 9.4.0 **runmqktool**

- Provides functions that are similar to those of the Java **keytool** command.
- Supports PKCS #12, JKS, and JCEKS key repositories.
- Requires that the IBM MQ Java runtime environment (JRE) component is installed.

If you need to manage certificates in a way that is FIPS-compliant, use the **runmqakm** command.

For more information about the **runmqakm** command, see runmqakm.

V 9.4.0 V 9.4.0 For more information about the **runmqktool** command, see runmqktool.

The topics in this section contain examples of how these commands are used to complete common certificate management tasks.

ALW Creating a self-signed personal certificate on AIX, Linux, and Windows

Follow this procedure to create a self-signed personal certificate in a key repository.

Note: IBM MQ does not support SHA-3 or SHA-5 algorithms. You can use the digital signature algorithm names SHA384WithRSA and SHA512WithRSA because both algorithms are members of the SHA-2 family.

Deprecated The digital signature algorithm names SHA3WithRSA and SHA5WithRSA are deprecated because they are an abbreviated form of SHA384WithRSA and SHA512WithRSA respectively.

You can create a self-signed certificate by using the **runmqakm** (GSKCapiCmd) or **runmqktool** (keytool) commands. If you need to manage SSL or TLS certificates in a way that is FIPS-compliant, use the **runmqakm** command.

For more information about why you might want to use self-signed certificates, see Using self-signed certificates for mutual authentication of two queue managers.

Not all digital certificates can be used with all CipherSpecs. Ensure that you create a certificate that is compatible with the CipherSpecs that you use. IBM MQ supports three different types of CipherSpec. For more information, see "Interoperability of Elliptic Curve and RSA CipherSpecs" on page 48.

To use the Type 1 CipherSpecs (those with names beginning ECDHE_ECDSA_) you must use the **runmqakm** command to create the certificate and you must specify an Elliptic Curve ECDSA signature algorithm parameter. For example, by specifying the parameter **-sig_alg** `EC_ecdsa_with_SHA384`.

Using runmqakm

Issue the following command to create a self-signed personal certificate with the **runmqakm** command:

```
runmqakm -cert -create -db filename -pw password -label label
         -dn distinguished_name -size key_size
         -x509version version -expire days -fips -sig_alg algorithm
```

where:

-db *filename*
: Specifies the fully qualified file name of the key repository. The key repository must already exist.

-pw *password*
: Specifies the password for the key repository.

-label *label*
: Specifies the certificate label. The certificate label is case-sensitive.

  The label of a TLS certificate that is used by IBM MQ is either the value of the **CERTLABL** attribute if it is set, or the default `ibmwebspheremq` with the name of the queue manager or the IBM MQ

MQI client user ID appended, all in lowercase. For more information, see "Digital certificate labels, understanding the requirements" on page 27.

-dn *distinguished_name*
Specifies the X.500 distinguished name enclosed in double quotation marks. At least one attribute is required in the distinguished name. You can supply multiple OU and DC attributes.

Note: The **runmqakm** command refers to the postal code attribute as `POSTALCODE`, not PC. Always specify `POSTALCODE` in the **-dn** parameter when you use the **runmqakm** command to request certificates with a postal code.

-size *key_size*
Specifies the key size. The value can be `512`, `1024`, or `2048`.

-x509version *version*
The version of X.509 certificate to create. The value can be 1, 2, or 3. The default is 3.

-expire *days*
The expiration time in days of the certificate. The default is 365 days for a certificate.

-fips
Specifies that the command is run in FIPS mode. Only the FIPS IBM Crypto for C (ICC) component is used and this component must be successfully initialized in FIPS mode. When in FIPS mode, the ICC component uses algorithms that have been FIPS 140-2 validated. If the ICC component does not initialize in FIPS mode, the **runmqakm** command fails.

-sig_alg
Specifies the hashing algorithm that is used when the certificate is created. This hashing algorithm is used to create the signature that is associated with the certificate. The value can be md5, `MD5_WITH_RSA`, `MD5WithRSA`, `SHA_WITH_DSA`, `SHA_WITH_RSA`, `sha1`, `SHA1WithDSA`, `SHA1WithECDSA`, `SHA1WithRSA`, `sha224`, `SHA224_WITH_RSA`, `SHA224WithDSA`, `SHA224WithECDSA`, `SHA224WithRSA`, `sha256`, `SHA256_WITH_RSA`, `SHA256WithDSA`, `SHA256WithECDSA`, `SHA256WithRSA`, `SHA2WithRSA`, `sha384`, `SHA384_WITH_RSA`, `SHA384WithECDSA`, `SHA384WithRSA`, `sha512`, `SHA512_WITH_RSA`, `SHA512WithECDSA`, `SHA512WithRSA`, `SHAWithDSA`, `SHAWithRSA`, `EC_ecdsa_with_SHA1`, `EC_ecdsa_with_SHA224`, `EC_ecdsa_with_SHA256`, `EC_ecdsa_with_SHA384`, or `EC_ecdsa_with_SHA512`.

The default value is `SHA1WithRSA`.

For more information about these parameters and the values that can be specified, see runmqakm -cert.

Using `runmqktool`

V 9.4.0 V 9.4.0

Issue the following command to create a self-signed personal certificate with the **runmqktool** command:

```
runmqktool -genkeypair -keystore filename -storepass password -storetype store_type
           -alias label -dname distinguished_name -validity days
           -keyalg key_algorithm -keysize key_size -sigalg signature_algorithm
```

where:

-keystore *filename*
Specifies the name of the key repository. The key repository is created if it does not exist.

-storepass *password*
Specifies the key repository password.

-storetype *store_type*
Specifies the key repository type.

-alias *label*
Specifies the certificate label. The certificate label is converted to lowercase.

-dname *distinguished_name*
Specifies the X.500 Distinguished Name for the certificate enclosed in double quotation marks.

-validity *days*
: Specifies the number of days for which the certificate is valid.

-keyalg *key_algorithm*
: Specifies the algorithm that is used to create the key pair.

-keysize *key_size*
: Specifies the key size.

-sigalg *signature_algorithm*
: Specifies the algorithm that is used to sign the certificate. For more information about the signature algorithms that can be specified, see Signature algorithms.

For more information about these parameters and the values that can be specified, see genkeypair.

ALW Requesting a personal certificate on AIX, Linux, and Windows

Follow this procedure to create a request for a personal certificate.

Note: IBM MQ does not support SHA-3 or SHA-5 algorithms. You can use the digital signature algorithm names SHA384WithRSA and SHA512WithRSA because both algorithms are members of the SHA-2 family.

Deprecated The digital signature algorithm names SHA3WithRSA and SHA5WithRSA are deprecated because they are an abbreviated form of SHA384WithRSA and SHA512WithRSA respectively.

You can request a personal certificate by using the **runmqakm** (GSKCapiCmd) or **runmqktool** (keytool) commands. If you need to manage SSL or TLS certificates in a way that is FIPS-compliant, use the **runmqakm** command.

Not all digital certificates can be used with all CipherSpecs. Ensure that you create a certificate that is compatible with the CipherSpecs that you use. IBM MQ supports three different types of CipherSpec. For more information, see "Interoperability of Elliptic Curve and RSA CipherSpecs" on page 48.

To use the Type 1 CipherSpecs (those with names beginning ECDHE_ECDSA_) you must use the **runmqakm** command to create the certificate and you must specify an Elliptic Curve ECDSA signature algorithm parameter. For example, by specifying the parameter **-sig_alg** EC_ecdsa_with_SHA384.

If you are using cryptographic hardware, see "Requesting a personal certificate for your PKCS #11 hardware" on page 549.

Using runmqakm

Issue the following command to create a certificate request with the **runmqakm** command:

```
runmqakm -certreq -create -db filename -pw password -label label
         -dn distinguished_name -size key_size
         -file filename -fips -sig_alg algorithm
```

where:

-db *filename*
: Specifies the fully qualified file name of a key repository. The key repository must already exist.

-pw *password*
: Specifies the password for the key repository.

-label *label*
: Specifies the certificate label. The certificate label is case-sensitive.

  The label of a TLS certificate that is used by IBM MQ is either the value of the **CERTLABL** attribute if it is set, or the default ibmwebspheremq with the name of the queue manager or the IBM MQ MQI client user ID appended, all in lowercase. For more information, see "Digital certificate labels, understanding the requirements" on page 27.

-dn *distinguished_name*
: Specifies the X.500 distinguished name enclosed in double quotation marks. At least one attribute is required in the distinguished name. You can supply multiple OU and DC attributes.

Note: The **runmqakm** command refers to the postal code attribute as POSTALCODE, not PC. Always specify POSTALCODE in the **-dn** parameter when you use the **runmqakm** command to request certificates with a postal code.

-size *key_size*
Specifies the key size. The value can be 512, 1024, or 2048.

-file *filename*
Specifies the file name for the certificate request.

-fips
Specifies that the command is run in FIPS mode. When in FIPS mode, the IBM Crypto for C (ICC) component uses algorithms that are FIPS 140-2 validated. If the ICC component does not initialize in FIPS mode, the **runmqakm** command fails.

-sig_alg
Specifies the hashing algorithm that is used when the certificate request is created. This hashing algorithm is used to create the signature that is associated with the certificate request. The value can be md5, MD5_WITH_RSA, MD5WithRSA, SHA_WITH_DSA, SHA_WITH_RSA, sha1, SHA1WithDSA, SHA1WithECDSA, SHA1WithRSA, sha224, SHA224_WITH_RSA, SHA224WithDSA, SHA224WithECDSA, SHA224WithRSA, sha256, SHA256_WITH_RSA, SHA256WithDSA, SHA256WithECDSA, SHA256WithRSA, SHA2WithRSA, sha384, SHA384_WITH_RSA, SHA384WithECDSA, SHA384WithRSA, sha512, SHA512_WITH_RSA, SHA512WithECDSA, SHA512WithRSA, SHAWithDSA, SHAWithRSA, EC_ecdsa_with_SHA1, EC_ecdsa_with_SHA224, EC_ecdsa_with_SHA256, EC_ecdsa_with_SHA384, or EC_ecdsa_with_SHA512.

The default value is SHA1WithRSA.

For more information about these parameters and the values that can be specified, see runmqakm -certreq.

Using runmqktool

V 9.4.0 V 9.4.0

Before you can create a certificate request with the **runmqktool** command, you must generate a key pair by using the **runmqktool -genkeypair** command. For more information about the **runmqktool -genkeypair** command, see "Creating a self-signed personal certificate on AIX, Linux, and Windows" on page 529.

Issue the following command to create a certificate request with the **runmqktool** command:

```
runmqktool -certreq -keystore filename -storepass password -alias label
           -file filename
```

where:

-keystore *filename*
Specifies the name of the key repository.

-storepass *password*
Specifies the key repository password.

-alias *label*
Specifies the certificate label. This is the certificate label that was specified when the key pair was generated. The certificate label is case-insensitive.

-file *filename*
Specifies the file name for the certificate request.

For more information about these parameters and the values that can be specified, see certreq.

What to do next

Submit a certificate request to a CA. When you receive the signed certificate from the CA, add the signed certificate into the key repository. For more information, see "Receiving personal certificates into a key repository on AIX, Linux, and Windows" on page 533.

ALW Renewing an existing personal certificate on AIX, Linux, and Windows

A personal certificate has an expiry date, after which the certificate can no longer be used. Follow this procedure to renew a personal certificate before it expires.

You can renew a personal certificate by using the **runmqakm** (GSKCapiCmd) command.

If you have a requirement to use larger key sizes for your personal certificates, you cannot renew an existing certificate. You must replace your existing key by following the steps described in "Requesting a personal certificate on AIX, Linux, and Windows" on page 531 to create a new certificate request that uses the key sizes you require.

Using runmqakm

Issue the following command to create a certificate request to renew a personal certificate with the **runmqakm** command:

```
runmqakm -certreq -recreate -db filename -pw password
         -label label -target filename
```

where:

-db *filename*
Specifies the fully qualified file name of the key repository.

-pw *password*
Specifies the password for the key repository.

-label *label*
Specifies the certificate label. The certificate label is case-sensitive.

-target *filename*
Specifies the file name for the certificate request.

What to do next

Submit a certificate request to a CA. When you receive the signed certificate from the CA, add the signed certificate into the key repository. For more information, see "Receiving personal certificates into a key repository on AIX, Linux, and Windows" on page 533.

ALW Receiving personal certificates into a key repository on AIX, Linux, and Windows

Use this procedure to receive a personal certificate into the key repository.

After the certificate authority (CA) sends you a new personal certificate, add it to the key repository from which you generated the new certificate request. If the CA sends the certificate as part of an email message, copy the certificate into a separate file.

Before you add the CA-signed personal certificate to the key repository, complete the steps in "Adding a CA certificate, or the public part of a trusted certificate, into a key repository on AIX, Linux, and Windows" on page 537 to add the CA certificate to the key repository.

You can receive a personal certificate into a key repository by using the **runmqakm** (GSKCapiCmd) or **runmqktool** (keytool) commands. If you need to manage SSL or TLS certificates in a way that is FIPS-compliant, use the **runmqakm** command.

If you are using cryptographic hardware, refer to “Receiving a personal certificate into your PKCS #11 hardware” on page 550.

Using `runmqakm`

Issue the following command to add a personal certificate to a key repository with the **runmqakm** command:

```
runmqakm -cert -receive -file filename -format format
         -db filename -pw password -fips
```

where:

-file *filename*
: Specifies the fully qualified file name of the personal certificate.

-db *filename*
: Specifies the fully qualified file name of the key repository. The key repository must already exist, and must be the same repository where you created the certificate request.

-pw *password*
: Specifies the password for the key repository.

-format *format*
: Specifies the format of the certificate. The value can be `ascii` for Base64-encoded ASCII or `binary` for binary DER data. The default is `ascii`.

-fips
: Specifies that the command is run in FIPS mode. When in FIPS mode, the IBM Crypto for C (ICC) component uses algorithms that have been FIPS 140-2 validated. If the ICC component does not initialize in FIPS mode, the **runmqakm** command fails.

For more information about these parameters and the values that can be specified, see runmqakm -cert.

Using `runmqktool`

V 9.4.0 V 9.4.0

Issue the following command to add a personal certificate to a key repository with the **runmqktool** command:

```
runmqktool -importcert -keystore filename -storepass password
           -alias label -file filename
```

where:

-keystore *filename*
: Specifies the fully qualified file name of the key repository. The key repository must already exist, and must be the same repository where you created the certificate request.

-storepass *password*
: Specifies the password for the key repository.

-alias *label*
: Specifies the label of the certificate that was used to create the certificate request. The certificate label is converted to lowercase.

-file *filename*
: Specifies the fully qualified file name of the personal certificate.

For more information about these parameters and the values that can be specified, see importcert.

What to do next

If the certificate is added to the queue manager's TLS key repository, issue the MQSC command **REFRESH SECURITY TYPE(SSL)** to refresh the queue manager's TLS key repository cache.

ALW Extracting a CA certificate from a key repository on AIX, Linux, and Windows

Follow this procedure to extract a certificate authority (CA) certificate from a key repository.

You can extract a CA certificate from a key repository by using the **runmqakm** (GSKCapiCmd) or **runmqktool** (keytool) commands. If you need to manage SSL or TLS certificates in a way that is FIPS-compliant, use the **runmqakm** command.

Using runmqakm

Issue the following command to extract a CA certificate with the **runmqakm** command:

```
runmqakm -cert -extract -db filename -pw password -label label
         -target filename -format format -fips
```

where:

-db *filename*
Specifies the fully qualified file name of the key repository.

-pw *password*
Specifies the password for the key repository.

-label *label*
Specifies the label of the CA certificate. The certificate label is case-sensitive.

-target *filename*
Specifies the fully qualified file name of the destination file.

-format *format*
Specifies the format of the certificate. The value can be `ascii` for Base64-encoded ASCII or `binary` for binary DER data. The default is `ascii`.

-fips
Specifies that the command is run in FIPS mode. When in FIPS mode, the IBM Crypto for C (ICC) component uses algorithms that have been FIPS 140-2 validated. If the ICC component does not initialize in FIPS mode, the **runmqakm** command fails.

For more information about these parameters and the values that can be specified, see runmqakm -cert.

Using runmqktool

V 9.4.0 V 9.4.0

Issue the following command to extract a CA certificate with the **runmqktool** command:

```
runmqktool -exportcert -keystore filename -storepass filename -alias label
           -file filename -rfc
```

where:

-keystore *filename*
Specifies the fully qualified file name of the key repository.

-storepass *password*
Specifies the password for the key repository.

-alias *label*
Specifies the label of the CA certificate. The certificate label is case-insensitive.

-file *filename*
Specifies the fully qualified file name of the destination file.

-rfc
Specifies that the output file is in Base64-encoded ASCII format, as defined by the Internet RFC 1421 standard. If this option is not specified, the output file is in binary format.

For more information about these parameters and the values that can be specified, see exportcert.

ALW Extracting the public part of a self-signed certificate from a key repository on AIX, Linux, and Windows

Follow this procedure to extract the public part of a self-signed certificate from a key repository.

You can extract the public part of a certificate from a key repository by using the **runmqakm** (GSKCapiCmd) or **runmqktool** (keytool) commands. If you need to manage SSL or TLS certificates in a way that is FIPS-compliant, use the **runmqakm** command.

Using runmqakm

Issue the following command to extract the public part of a self-signed certificate with the **runmqakm** command:

```
runmqakm -cert -extract -db filename -pw password -label label
            -target filename -format format -fips
```

where:

-db *filename*
Specifies the fully qualified file name of the key repository.

-pw *password*
Specifies the password for the key repository.

-label *label*
Specifies the label of the CA certificate. The certificate label is case-sensitive.

-target *filename*
Specifies the fully qualified file name of the destination file.

-format *format*
Specifies the format of the certificate. The value can be `ascii` for Base64-encoded ASCII or `binary` for binary DER data. The default is `ascii`.

-fips
Specifies that the command is run in FIPS mode. When in FIPS mode, the IBM Crypto for C (ICC) component uses algorithms that have been FIPS 140-2 validated. If the ICC component does not initialize in FIPS mode, the **runmqakm** command fails.

For more information about these parameters and the values that can be specified, see runmqakm -cert.

Using runmqktool

V 9.4.0 V 9.4.0

Issue the following command to extract the public part of a self-signed certificate with the **runmqktool** command:

```
runmqktool -exportcert -keystore filename -storepass filename -alias label
           -file filename -rfc
```

where:

-keystore *filename*
Specifies the fully qualified file name of the key repository.

-storepass *password*
Specifies the password for the key repository.

-alias *label*
Specifies the label of the CA certificate. The certificate label is case-insensitive.

-file *filename*
Specifies the fully qualified file name of the destination file.

-rfc
Specifies that the output file is in Base64-encoded ASCII format, as defined by the Internet RFC 1421 standard. If this option is not specified, the output file is in binary format.

For more information about these parameters and the values that can be specified, see exportcert.

ALW Adding a CA certificate, or the public part of a trusted certificate, into a key repository on AIX, Linux, and Windows

Follow this procedure to add a CA certificate or the public part of a trusted certificate to a key repository.

You can add a CA certificate, or the public part of a trusted certificate, into a key repository by using the **runmqakm** (GSKCapiCmd) or **runmqktool** (keytool) commands. If you need to manage SSL or TLS certificates in a way that is FIPS-compliant, use the **runmqakm** command.

If the certificate that you want to add is in a certificate chain, you must also add all the certificates that are above it in the chain. You must add the certificates in strictly descending order starting from the root, followed by the CA certificate immediately below it in the chain, and so on.

Note:

- Ensure that the certificate is in ASCII (UTF-8) or binary (DER) encoding.
- Due to a restriction in the IBM Java 8 **keytool** command, **runmqktool** cannot import certificates in printable encoding format (also known as Base64 encoding) as defined by the Internet RFC 1421 if the file contains comments. To import a certificate in printable encoding format, remove all comments from the file. The file must begin with a string that starts with "-----BEGIN", and end with a string that starts with "-----END".

Using runmqakm

Issue the following command to add a trusted certificate to a key repository with the **runmqakm** command:

```
runmqakm -cert -add -db filename -pw password -label label
         -file filename -format ascii -fips
```

where:

-db *filename*
Specifies the fully qualified file name of the key repository. The key repository must already exist.

-pw *password*
Specifies the password for the key repository.

-label *label*
Specifies the certificate label. The certificate label is case-sensitive.

-file *filename*
Specifies the name of the file containing the certificate.

-format *ascii*
Specifies the format of the certificate. The value can be `ascii` for Base64-encoded ASCII or `binary` for binary DER data. The default is `ascii`.

-fips
Specifies that the command is run in FIPS mode. When in FIPS mode, the IBM Crypto for C (ICC) component uses algorithms that have been FIPS 140-2 validated. If the ICC component does not initialize in FIPS mode, the **runmqakm** command fails.

For more information about these parameters and the values that can be specified, see runmqakm -cert.

Using **runmqktool**

V 9.4.0 V 9.4.0

Issue the following command to add a trusted certificate to a key repository with the **runmqktool** command:

```
runmqktool -importcert -keystore filename -storepass password
           -alias label -file filename
```

where:

-keystore *filename*
  Specifies the fully qualified file name of the key repository. The key repository is created if it does not exist.

-storepass *password*
  Specifies the password for the key repository.

-alias *label*
  Specifies the certificate label. The certificate label is converted to lowercase.

-file *filename*
  Specifies the fully qualified file name of the personal certificate.

For more information about these parameters and the values that can be specified, see importcert.

ALW Exporting a personal certificate from a key repository on AIX, Linux, and Windows

Follow this procedure export a personal certificate from a key repository.

Exporting a certificate copies the certificate and its associated public and private keys into another key repository.

You can export a certificate from a key repository by using the **runmqakm** (GSKCapiCmd) or **runmqktool** (keytool) commands. If you need to manage SSL or TLS certificates in a way that is FIPS-compliant, use the **runmqakm** command.

Using **runmqakm**

Issue the following command to export a certificate with the **runmqakm** command:

```
runmqakm -cert -export -db filename -pw password -label label
         -target filename -target_pw password -target_type type
         -encryption strength -fips
```

where:

-db *filename*
  Specifies the fully qualified file name of the key repository that contains the certificate.

-pw *password*
  Specifies the password for the key repository that contains the certificate.

-label *label*
  Specifies the label of the certificate to export. The certificate label is case-sensitive.

-target *filename*
  Specifies the fully qualified file name of the destination key repository. The key repository is created if it does not exist.

-target_pw *password*
  Specifies the password for the destination key repository.

-target_type *type*
  Specifies the type of the destination key repository. The value can be `cms` or `pkcs12`. The default is `cms`.

-encryption *strength*
Specifies the strength of encryption that is used in certificate export command. The value can be `strong` or `weak`. The default is `strong`.

-fips
Specifies that the command is run in FIPS mode. When in FIPS mode, the IBM Crypto for C (ICC) component uses algorithms that have been FIPS 140-2 validated. If the ICC component does not initialize in FIPS mode, the **runmqakm** command fails.

For more information about these parameters and the values that can be specified, see runmqakm -cert.

Using runmqktool

V 9.4.0 V 9.4.0

Issue the following command to export a certificate with the **runmqktool** command:

```
runmqktool -importkeystore -srckeystore filename -srcstorepass password
           -destkeystore filename -deststoretype type
           -deststorepass password -destkeypass password
           -srcalias label -destalias label
```

where:

-srckeystore *filename*
Specifies the fully qualified file name of the key repository that contains the certificate.

-srcstorepass *password*
Specifies the password for the key repository that contains the certificate.

-destkeystore *filename*
Specifies the fully qualified file name of the destination key repository. The key repository is created if it does not exist.

-deststorepass *password*
Specifies the password for the destination key repository.

-destkeypass *password*
Specifies the password to protect the key in the destination key repository. If this parameter is not specified, the key is protected with the password that is used to protect the key in the source key repository.

-deststoretype *type*
Specifies the type of the destination key repository.

-srcalias *label*
Specifies the label of the certificate to export. The certificate label is case-insensitive.

-destalias *label*
Specifies the label of the certificate in the destination key repository. If this parameter is not specified, the same label is assigned to the certificate as in the source key repository.

The certificate label is converted to lowercase.

-file *filename*
Specifies the fully qualified file name of the destination file.

For more information about these parameters and the values that can be specified, see importkeystore.

ALW Importing a personal certificate into a key repository on AIX, Linux, and Windows

Follow this procedure to import a personal certificate into a key repository.

Importing a certificate copies the certificate and its associated public and private keys from one key repository to another key repository.

Before importing a personal certificate to a key repository, you must first add the full valid chain of issuing CA certificates to the key repository. For more information, see "Adding a CA certificate, or the public part of a trusted certificate, into a key repository on AIX, Linux, and Windows" on page 537.

You can import a certificate to a key repository by using the **runmqakm** (GSKCapiCmd) or **runmqktool** (keytool) commands. If you need to manage SSL or TLS certificates in a way that is FIPS-compliant, use the **runmqakm** command.

Using runmqakm

Issue the following command to import a certificate with the **runmqakm** command:

```
runmqakm -cert -import -file filename -pw password -type type
         -target filename -target_pw password -target_type type
         -label label -new_label label -fips
```

where:

-file *filename*
Specifies the fully qualified file name of the key repository that contains the certificate.

-pw *password*
Specifies the password for the key repository that contains the certificate.

-type *type*
Specifies the type of the key repository that contains the certificate. The value can be `cms` or `pkcs12`. The default is `cms`.

-target *filename*
Specifies the fully qualified file name of the destination key repository. The key repository is created if it does not exist.

-target_pw *password*
Specifies the password for the destination key repository.

-target_type *type*
Specifies the type of the destination key repository. The value can be `cms` or `pkcs12`. The default is `cms`.

-label *label*
Specifies the label of the certificate to import from the source key repository. The certificate label is case-sensitive.

-new_label *label*
Specifies the label that is assigned to the certificate in the target key repository. If this parameter is not specified, the same label is assigned to the certificate as in the source key repository.

-fips
Specifies that the command is run in FIPS mode. When in FIPS mode, the IBM Crypto for C (ICC) component uses algorithms that have been FIPS 140-2 validated. If the ICC component does not initialize in FIPS mode, the **runmqakm** command fails.

For more information about these parameters and the values that can be specified, see runmqakm -cert.

Using runmqktool

V 9.4.0 V 9.4.0

Issue the following command to import a certificate with the **runmqktool** command:

```
runmqktool -importkeystore -srckeystore filename -srcstorepass password
           -destkeystore filename -deststoretype type
           -deststorepass password -destkeypass password
           -srcalias label -destalias label
```

where:

-srckeystore *filename*
Specifies the fully qualified file name of the key repository that contains the certificate.

-srcstorepass *password*
Specifies the password for the key repository that contains the certificate.

-destkeystore *filename*
Specifies the fully qualified file name of the destination key repository. The key repository is created if it does not exist.

-deststorepass *password*
Specifies the password for the destination key repository.

-destkeypass *password*
Specifies the password to protect the key in the destination key repository. If this parameter is not specified, the key is protected with the password that is used to protect the key in the source key repository.

Note: For a PKCS #12 key repository, the key must be protected with the same password as the destination key repository.

-deststoretype *type*
Specifies the type of the destination key repository.

-srcalias *label*
Specifies the label of the certificate to in the source key repository. The certificate label is case-insensitive.

-destalias *label*
Specifies the label of the certificate in the destination key repository. If this parameter is not specified, the same label is assigned to the certificate as in the source key repository.

The certificate label is converted to lower case.

-file *filename*
Specifies the fully qualified file name of the destination file.

For more information about these parameters and the values that can be specified, see importkeystore.

ALW Importing a personal certificate from a Microsoft .pfx file

Follow this procedure to import a certificate from a Microsoft .pfx file on AIX, Linux, and Windows.

A .pfx file can contain two certificates relating to the same key. One is a personal or site certificate that contains both a public and private key. The other is a CA (signer) certificate that contains only a public key. These certificates cannot coexist in the same CMS key repository, so only one of them can be imported.

The certificate label is attached to only the signer certificate. The personal certificate is identified by a system generated Unique User Identifier (UUID). Follow this procedure to import a personal certificate from a .pfx file and set the personal certificate label to the label that is assigned to the CA certificate in the .pfx file. The issuing CA certificates should already be added to the target key database.

Using `runmqakm`

Issue the following command to import a certificate from a .pfx file with the **`runmqakm`** command:

```
runmqakm -cert -import -file filename -pw password -type pkcs12
         -target filename -target_pw password -target_type type
         -label label -new_label label -fips -pfx
```

where:

-file *filename*
Specifies the fully qualified name of the .pfx file.

-pw *password*
Specifies the password for the .pfx file.

-type pkcs12
Specifies the type of the key repository.

-target *filename*
Specifies the fully qualified file name of the destination key repository. The key repository is created if it does not exist.

-target_pw *password*
Specifies the password for the destination key repository.

-target_type *type*
Specifies the type of the destination key repository. The value can be `cms` or `pkcs12`. The default is `cms`.

-label *label*
Specifies the label of the certificate to import from the source key repository. The certificate label is case-sensitive.

-new_label *label*
Specifies the label that is assigned to the certificate in the target key repository. If this parameter is not specified, the same label is assigned to the certificate as in the source key repository.

-fips
Specifies that the command is run in FIPS mode. When in FIPS mode, the IBM Crypto for C (ICC) component uses algorithms that have been FIPS 140-2 validated. If the ICC component does not initialize in FIPS mode, the **runmqakm** command fails.

-pfx
Indicates that the source key repository uses the PFX format.

For more information about these parameters and the values that can be specified, see runmqakm -cert.

ALW Importing a personal certificate from a PKCS #7 file

Follow this procedure to import a certificate from a PKCS #7 file on AIX, Linux, and Windows.

Use the **runmqakm** command to import certificates from a PKCS #7 file on AIX, Linux, and Windows.

Adding a CA certificate or the public part of a trusted certificate

Issue the following command to add a CA certificate, or the public part of a trusted certificate, from a PKCS #7 file:

```
runmqakm -cert -add -db filename -pw password -type type
         -label label -file filename
```

where:

-db *filename*
Specifies the fully qualified name of the key repository.

-pw *password*
Specifies the password for the key repository.

-type *type*
Specifies the type of the key repository.

-label *label*
Specifies the label of the certificate to add. The certificate label is case-sensitive.

The label is assigned to the first certificate that is added. All other certificates, if present, are labeled with their subject name.

-file *filename*
Specifies the fully qualified name of the PKCS #7 file.

For more information about these parameters and the values that can be specified, see runmqakm -cert.

Importing a personal certificate

Issue the following command to import a personal certificate from a PKCS #7 file:

```
runmqakm -cert -import -file filename -pw password -type pkcs7
         -target filename -target_pw password -target_type type
         -label label -new_label label
```

where:

-file *filename*
Specifies the fully qualified name of the PKCS #7 file.

-pw *password*
Specifies the password for the PKCS #7 file.

-type pkcs7
Specifies the type of the PKCS #7 file.

-target *filename*
Specifies the fully qualified file name of the destination key repository. The key repository is created if it does not exist.

-target_pw *password*
Specifies the password for the destination key repository.

-target_type *type*
Specifies the type of the destination key repository. The value can be `cms` or `pkcs12`. The default is `cms`.

-label *label*
Specifies the label of the certificate to import from the PKCS #7 file. The certificate label is case-sensitive.

-new_label *label*
Specifies the label that is assigned to the certificate in the target key repository. If this parameter is not specified, the same label is assigned to the certificate as in the source key repository.

For more information about these parameters and the values that can be specified, see runmqakm -cert.

ALW Listing the certificates in a key repository on AIX, Linux, and Windows

Use this procedure to list the certificates that are in a key repository.

You can display information about the certificates that are in a key repository by using the **runmqakm** (GSKCapiCmd) or **runmqktool** (keytool) commands.

Using runmqakm

- Issue the following command to list the labels of the certificates in a key repository with the **runmqakm** command:

  ```
  runmqakm -cert -list -db filename -pw password
  ```

- Issue the following command to list the details of a certificate in a key repository with the **runmqakm** command:

  ```
  runmqakm -cert -details -showOID -db filename -pw password
           -label label
  ```

where:

-file *filename*
Specifies the fully qualified file name of the key repository.

-pw *password*
Specifies the password for the key repository.

-label ***label***
  Specifies the label of the certificate to list. The certificate label is case-sensitive.

For more information about these parameters and the values that can be specified, see runmqakm -cert.

Using `runmqktool`

V 9.4.0 V 9.4.0

- Issue the following command to list the labels of the certificates in a key repository with the **`runmqktool`** command:

  ```
  runmqktool -list -keystore filename -storepass password
  ```

- Issue the following command to list the details of a certificate in a key repository with the **`runmqktool`** command:

  ```
  runmqktool -list -keystore filename -storepass password -alias label -v
  ```

where:

-keystore ***filename***
  Specifies the fully qualified file name of the key repository.

-storepass ***password***
  Specifies the password for the key repository.

-alias ***label***
  Specifies the label of the certificate to list. The certificate label is case-insensitive.

-v
  Requests verbose output that includes the certificate details.

For more information about these parameters and the values that can be specified, see list.

ALW Deleting a certificate from a key repository on AIX, Linux, and Windows

Use this procedure to delete a personal or CA certificate from a key repository.

You can delete a certificate from a key repository by using the **`runmqakm`** (GSKCapiCmd) or **`runmqktool`** (keytool) commands. If you need to manage SSL or TLS certificates in a way that is FIPS-compliant, use the **`runmqakm`** command.

Using `runmqakm`

Issue the following command to delete a certificate with the **`runmqakm`** command:

```
runmqakm -cert -delete -db filename -pw password -label label -fips
```

where:

-file ***filename***
  Specifies the fully qualified file name of the key repository.

-pw ***password***
  Specifies the password for the key repository.

-label ***label***
  Specifies the label of the certificate to delete. The certificate label is case-sensitive.

-fips
  Specifies that the command is run in FIPS mode. When in FIPS mode, the IBM Crypto for C (ICC) component uses algorithms that have been FIPS 140-2 validated. If the ICC component does not initialize in FIPS mode, the **`runmqakm`** command fails.

For more information about these parameters and the values that can be specified, see runmqakm -cert.

Using **runmqktool**

V 9.4.0 V 9.4.0

Issue the following command to delete a certificate with the **runmqktool** command:

```
runmqktool -delete -keystore filename -storepass password -alias label
```

where:

-keystore *filename*
: Specifies the fully qualified file name of the key repository.

-storepass *password*
: Specifies the password for the key repository.

-alias *label*
: Specifies the label of the certificate to delete. The certificate label is case-insensitive.

For more information about these parameters and the values that can be specified, see delete.

ALW Converting a key repository on AIX, Linux, and Windows

Use this procedure to convert a key repository to a different type.

You can convert a key repository password to a different type by using the **runmqakm** (GSKCapiCmd) or **runmqktool** (keytool) commands.

Using **runmqakm**

Issue the following command to convert a key repository with the **runmqakm** command:

```
runmqakm -keydb -convert -db filename -pw password
         -new_db filename -new_pw password
         -old_format type -new_format type
```

where:

-file *filename*
: Specifies the fully qualified file name of the key repository.

-pw *password*
: Specifies the password for the key repository.

-new_db *filename*
: Specifies the fully qualified file name of the new key repository.

-new_pw *password*
: Specifies the password for the new key repository.

-old_format *type*
: Specifies the current type of the key repository. The following values can be specified:

- `pkcs12`
- `cms`

-new_format *type*
: Specifies the new type of the key repository. The following values can be specified:

- `pkcs12`
- `cms`

For more information about these parameters and the values that can be specified, see runmqakm -keydb.

Using **runmqktool**

V 9.4.0 V 9.4.0

Issue the following command to convert a key repository with the **runmqktool** command:

```
runmqktool -importkeystore -srckeystore filename -destkeystore filename
           -srcstoretype type -deststoretype type
           -srcstorepass password -deststorepass password
```

where:

-all
Specifies that the password is also changed for all entries that are protected with the same password as the key repository.

-keystore *filename*
Specifies the fully qualified file name of the key repository.

-destkeystore *filename*
Specifies the fully qualified file name of the new key repository.

-srcstoretype *type*
Specifies the key repository type.

-deststoretype *type*
Specifies the new key repository type.

-srcstorepass *password*
Specifies the password for the key repository.

-deststorepass *password*
Specifies the password for the new key repository.

For more information about these parameters and the values that can be specified, see importkeystore.

ALW Changing the key repository password on AIX, Linux, and Windows

Use this procedure to change the key repository password.

You can change the key repository password by using the **runmqakm** (GSKCapiCmd) or **runmqktool** (keytool) commands.

Note:

- V 9.4.0 V 9.4.0 The **runmqktool** command allows the key repository password to be changed independently of the passwords that protect individual private and secret keys. For PKCS #12 key repositories, the key repository password, and the passwords that protect all keys in the key repository, must be the same. If the **runmqktool** command is used to change the key repository password, ensure that the **-all** parameter is specified so that the key passwords are also changed.
- If the key repository password is not stored in a stash file, you must also change the password that is stored in the queue manager configuration or any IBM MQ client applications that access the key repository. For more information, see “Supplying the key repository password for a queue manager on AIX, Linux, and Windows” on page 294 and “Supplying the key repository password for an IBM MQ MQI client on AIX, Linux, and Windows” on page 296.

Using runmqakm

Issue the following command to change the key repository password with the **runmqakm** command:

```
runmqakm -keydb -changepw -db filename -pw password -new_pw password -stash
```

where:

-file *filename*
Specifies the fully qualified file name of the key repository.

-pw *password*
Specifies the current password for the key repository.

-new_pw *password*
: Specifies the new password for the key repository.

-stash
: Optional. Specify this option to store the new key repository password in a stash file. You do not need to store the password in a stash file if you encrypt the password by using the IBM MQ password protection system instead.

For more information about these parameters and the values that can be specified, see runmqakm -keydb.

Using `runmqktool`

V 9.4.0 V 9.4.0

Issue the following command to change the key repository password with the **`runmqktool`** command:

```
runmqktool -storepasswd -all -keystore filename -storepass password
           -new password
```

where:

-all
: Specifies that the password is also changed for all entries that are protected with the same password as the key repository.

-keystore *filename*
: Specifies the fully qualified file name of the key repository.

-storepass *password*
: Specifies the current password for the key repository.

-new *password*
: Specifies the new password for the key repository.

For more information about these parameters and the values that can be specified, see storepasswd.

ALW Managing secret keys on AIX, Linux, and Windows

Follow this procedure to manage secret keys in a key repository.

You can manage secret keys by using the **`runmqakm`** (GSKCapiCmd) command. Secret keys that are generated by using the **`runmqktool`** (keytool) command cannot be used with IBM MQ.

Creating a secret key

Issue the following command to create a random secret key with the **`runmqakm`** command:

```
runmqakm -secretkey -create -db filename -pw password
         -label label -size key_size
```

where:

-db *filename*
: Specifies the fully qualified file name of the key repository. The key repository must already exist.

-pw *password*
: Specifies the password for the key repository.

-label *label*
: Specifies the label that is attached to the key.

-size *key_size*
: Specifies the key size in bytes.

For more information about these parameters and the values that can be specified, see runmqakm -secretkey.

Extracting a secret key

Issue the following command to extract a secret key with the **runmqakm** command:

```
runmqakm -secretkey -extract -db filename -pw password
         -label label -target filename -format format
```

where:

-db *filename*
Specifies the fully qualified file name of the key repository. The key repository must already exist.

-pw *password*
Specifies the password for the key repository.

-label *label*
Specifies the label of the key to extract.

-target *filename*
Specifies the fully qualified file name of the destination file.

-format *format*
Specifies the format of the key in the destination file. The value can be `ascii` for Base64-encoded ASCII or `binary` for a binary copy of the key. The default is `ascii`.

For more information about these parameters and the values that can be specified, see runmqakm -secretkey.

Adding a secret key

Issue the following command to extract a secret key with the **runmqakm** command:

```
runmqakm -secretkey -add -db filename -pw password
         -label label -file filename -format format
```

where:

-db *filename*
Specifies the fully qualified file name of the key repository. The key repository must already exist.

-pw *password*
Specifies the password for the key repository.

-label *label*
Specifies the label that is attached to the key.

-file *filename*
Specifies the name of the file containing the key.

-format *format*
Specifies the format of the key. The value can be `ascii` for Base64-encoded ASCII or `binary` for binary data. The default is `ascii`.

For more information about these parameters and the values that can be specified, see runmqakm -secretkey.

ALW Managing certificates on PKCS #11 hardware

You can manage digital certificates on cryptographic hardware that supports the PKCS #11 interface.

You must create a key repository to prepare the IBM MQ environment, even if you do not intend to store any certificates in it, but will store all your certificates on your cryptographic hardware. A key repository is necessary for the queue manager to reference in its **SSLKEYR** attribute, or for the client application to reference in the MQSSLKEYR environment variable. This key repository is also required if you are creating a certificate request.

Create the key repository by using the **runmqakm** (GSKCapiCmd) command.

Issue the following command to create a key repository with the **runmqakm** command:

```
runmqakm -keydb -create -db filename -pw password -type type -stash
```

where:

-db *filename*
Specifies the fully qualified file name of the key repository.

-pw *password*
Specifies the password for the key repository.

-type *type*
Specifies the type of database. The value must be cms or pkcs12 for a key repository that is used by IBM MQ.

-stash
Optional. If specified, the encrypted key repository password is saved to a file.

ALW *Requesting a personal certificate for your PKCS #11 hardware*

Use this procedure to request a personal certificate for either a queue manager or an IBM MQ MQI client with your cryptographic hardware.

Note: IBM MQ does not support SHA-3 or SHA-5 algorithms. You can use the digital signature algorithm names SHA384WithRSA and SHA512WithRSA because both algorithms are members of the SHA-2 family.

Deprecated The digital signature algorithm names SHA3WithRSA and SHA5WithRSA are deprecated because they are an abbreviated form of SHA384WithRSA and SHA512WithRSA respectively.

Before you create a certificate request in your cryptographic hardware, complete the steps that are described in "Managing certificates on PKCS #11 hardware" on page 548 to create a key repository.

Issue the following command to create a certificate request with the **runmqakm** (GSKCapiCmd) command:

```
runmqakm -certreq -create -crypto module_name -tokenlabel hardware_token
         -pw password -label label
         -dn distinguished_name -size key_size
         -file filename -fips -sig_alg algorithm
```

where:

-crypto *module_name*
Specifies the fully qualified name of the PKCS #11 library supplied with the cryptographic hardware.

-tokenlabel *hardware_token*
Specifies the PKCS #11 cryptographic device token label.

-pw *password*
Specifies the password to access the cryptographic hardware.

-label *label*
Specifies the certificate label.

The label of a TLS certificate that is used by IBM MQ is either the value of the **CERTLABL** attribute if it is set, or the default ibmwebspheremq with the name of the queue manager or the IBM MQ MQI client user ID appended, all in lowercase. For more information, see "Digital certificate labels, understanding the requirements" on page 27.

-dn *distinguished_name*
Specifies the X.500 distinguished name enclosed in double quotation marks. At least one attribute is required in the distinguished name. You can supply multiple OU and DC attributes.

Note: The **runmqakm** command refers to the postal code attribute as POSTALCODE, not PC. Always specify POSTALCODE in the **-dn** parameter when you use the **runmqakm** command to request certificates with a postal code.

-size *key_size*
Specifies the key size. The value can be 512, 1024, or 2048.

-file *filename*
Specifies the file name for the certificate request.

-fips
Specifies that the command is run in FIPS mode. When in FIPS mode, the IBM Crypto for C (ICC) component uses algorithms that are FIPS 140-2 validated. If the ICC component does not initialize in FIPS mode, the **runmqakm** command fails.

-sig_alg
Specifies the hashing algorithm that is used when the certificate request is created. This hashing algorithm is used to create the signature that is associated with the certificate request. The value can be md5, MD5_WITH_RSA, MD5WithRSA, SHA_WITH_DSA, SHA_WITH_RSA, sha1, SHA1WithDSA, SHA1WithECDSA, SHA1WithRSA, sha224, SHA224_WITH_RSA, SHA224WithDSA, SHA224WithECDSA, SHA224WithRSA, sha256, SHA256_WITH_RSA, SHA256WithDSA, SHA256WithECDSA, SHA256WithRSA, SHA2WithRSA, sha384, SHA384_WITH_RSA, SHA384WithECDSA, SHA384WithRSA, sha512, SHA512_WITH_RSA, SHA512WithECDSA, SHA512WithRSA, SHAWithDSA, SHAWithRSA, EC_ecdsa_with_SHA1, EC_ecdsa_with_SHA224, EC_ecdsa_with_SHA256, EC_ecdsa_with_SHA384, or EC_ecdsa_with_SHA512.

The default value is SHA1WithRSA.

For more information about these parameters and the values that can be specified, see runmqakm -certreq.

What to do next

Submit a certificate request to a CA. When you receive the signed certificate from the CA, add the signed certificate into the key repository. For more information, see "Receiving a personal certificate into your PKCS #11 hardware" on page 550.

ALW *Receiving a personal certificate into your PKCS #11 hardware*

Use this procedure to receive a personal certificate for either a queue manager or an IBM MQ MQI client to your cryptographic hardware.

Add the CA certificate of the CA that signed the personal certificate into either the cryptographic hardware or the secondary key repository. Do this before you receive the signed certificate into the cryptographic hardware. To add a CA certificate to a key repository file, follow the procedure in "Adding a CA certificate, or the public part of a trusted certificate, into a key repository on AIX, Linux, and Windows" on page 537.

Issue the following command to add a personal certificate to a key repository with the **runmqakm** (GSKCapiCmd) command:

```
runmqakm -cert -receive -file filename -crypto module_name
         -tokenlabel hardware_token -pw hardware_password
         -format cert_format -fips
         -secondaryDB filename -secondaryDBpw password
```

where:

-file *filename*
Specifies the fully qualified file name of the file containing the personal certificate.

-crypto *module_name*
Specifies the fully qualified name of the PKCS #11 library supplied with the cryptographic hardware.

-tokenlabel *hardware_token*
Specifies the PKCS #11 cryptographic device token label.

-pw *hardware_password*
Specifies the password to access the cryptographic hardware.

-format *cert_format*
: Specifies the format of the certificate. The value can be `ascii` for Base64-encoded ASCII or `binary` for binary DER data. The default is ASCII.

-fips
: Specifies that the command is run in FIPS mode. When in FIPS mode, the IBM Crypto for C (ICC) component uses algorithms that are FIPS 140-2 validated. If the ICC component does not initialize in FIPS mode, the **`runmqakm`** command fails.

-secondaryDB *filename*
: Specifies the fully qualified file name of the key repository file that is used to store the CA certificate.

-secondaryDBpw *password*
: Specifies the password for the key repository file that is used to store the CA certificate.

Protecting passwords in IBM MQ component configuration files

To use certain features of IBM MQ, you might need to supply passwords that are used by the feature. Passwords that are supplied to IBM MQ can be protected by using a password protection system.

The following list explains the terminology that is used for each component that processes encrypted passwords:

Initial key
: The encryption key that is used to protect the password.

Default initial key
: The default encryption key that is used if you do not supply an initial key when the password is encrypted.

Plain text string
: The string that is encrypted, commonly a password.

Encrypted password string
: A string that contains the encrypted password in a format that IBM MQ understands.

Specifying the initial key

For each component, you can choose to specify an initial key that is used to encrypt passwords.

- If you do not specify an initial key, the default initial key for the component is used. The default initial key is the same for all IBM MQ installations. This means that a password that is encrypted with the default initial key is not securely protected as it might be possible for a different installation to decrypt the password.
- If you provide your own, unique, initial key, only users with access to the initial key that you provide can decrypt the password.

Attention: To provide the highest level of security for stored passwords, supply a unique initial key for each IBM MQ component.

If you choose to use your own initial key, specify a unique initial key for each component that is listed. The initial key is used to protect any passwords that are stored in the configuration of that component. The same initial key must also be made available to the component for the password to be decrypted.

Most components require the initial key to be supplied in a file. The initial key that is contained in the initial key file must meet the following requirements:

- It must be at least one character long.
- It must be a single line of text.

The maximum length of the initial key is unlimited, and any characters can be specified. For adequate security, specify an initial key that is at least 16 characters long. For example, your initial key file might contain the following string:

```
Th1sIs@n3Ncypt|onK$y
```

Access to the initial key file must be limited to only the users who need to access the initial key by using the operating system file permissions.

For more information about the benefits and limitations of password protection, see "The limits to protection through password encryption" on page 558.

Protecting passwords in each IBM MQ component

Several IBM MQ components can protect stored passwords. Depending on the component, these passwords might be supplied by using one of the following mechanisms:

- Provided directly to the IBM MQ queue manager or IBM MQ client.
- Specified in an environment variable.
- Stored in a configuration file.

Each component provides a method to encrypt passwords. In most components, passwords must be encrypted before they are supplied to IBM MQ or stored in the configuration.

Important: An encrypted password that is generated for use with one component cannot be copied to the configuration file of another component. A password that is encrypted for use by a particular component must be protected with the utility that is provided by the same component.

Details of how to protect passwords for each IBM MQ component that supports password protection are listed in the following sections:

- Advanced Message Security
- "Managed File Transfer" on page 553
- "IBM MQ Internet Pass-Thru" on page 554
- "IBM MQ clients that use cryptographic hardware" on page 555
- "IBM MQ queue manager" on page 555
- "IBM MQ C client applications" on page 556
- V 9.4.0 "Native HA configurations" on page 556
- V 9.4.0 "IBM MQ queue manager (AuthToken stanza in the qm.ini file)" on page 557

Advanced Message Security

Advanced Message Security (AMS) Java clients require access to a keystore that contains the private keys that are used to protect messages.

Advanced Message Security (AMS) MQI clients or queue managers that are configured to perform MCA interception might require access to PKCS#11 cryptographic hardware, or PEM files that contain the private keys that are used to protect messages.

To access these key repositories, a password must be provided in the AMS configuration file that is called `keystore.conf`. Use the **`runamscred`** command to protect the sensitive information that is contained in the `keystore.conf` file. For example,

```
runamscred -f <keystore configuration file>
```

The **`runamscred`** command protects sensitive parameters within the file that is specified by using the **`-f`** parameter.

Two **`runamscred`** commands are available in an IBM MQ installation:

- An MQI **`runamscred`** command that is located in `<IBM MQ installation root>/bin`
- A Java **`runamscred`** command that is located in `<IBM MQ installation root>/java/bin`

Attention: To ensure compatibility,

1. Use the Java **runamscred** command to protect configuration files that are used with Java AMS clients and the MQI **runamscred** command to protect configuration files for IBM MQ MQI clients that use AMS.
2. Verify that all the necessary sensitive information is protected after you run the **runamscred** command.
3. Supply the file that contains the protected password as normal to AMS enabled applications.

By default, the **runamscred** command encrypts password in the configuration file with the default initial key. To encrypt the passwords with a specific initial key, use one of the following mechanisms to specify the name of the file that contains the initial key, in order of priority:

1. The **-sf** parameter to the **runamscred** command.
2. The **MQS_AMSCRED_KEYFILE** environment variable.
3. The **amscred.keyfile** parameter in the keystore.conf configuration file.

CAUTION: The default initial key is the same for all IBM MQ installations. To protect passwords securely, supply an initial key that is unique to your installation when you encrypt passwords.

If you specify an initial key file when you run the **runamscred** command to encrypt the passwords in the AMS configuration, you must also specify the same initial key file when AMS applications run. The following mechanisms can be used to specify the name of the initial key file, in order of priority:

1. The **MQS_AMSCRED_KEYFILE** environment variable.
2. The **amscred.keyfile** parameter in the keystore.conf configuration file.

By default, the **runamscred** command protects credentials with a protection system that is not compatible with AMS versions earlier than IBM MQ 9.2. To protect configuration files with the credentials protection system that is compatible with versions earlier than IBM MQ 9.2, specify the **-sp 0** parameter when the **runamscred** command is run.

Managed File Transfer

Managed File Transfer (MFT) stores credentials that are required to access queue managers and other resources in the following XML property files:

MQMFTCredentials.xml
This file contains the following credentials:

- Credentials that are used to connect to agent, coordination, and command queue managers.
- Passwords that are used to access keystores that are used for secure communications.

ProtocolBridgeCredentials.xml
This file contains credentials that are used to connect to Protocol Servers, such as FTP, SFTP, and FTPS.

ConnectDirectCredentials.xml
This file contains credentials that are used by a Connect:Direct® agent to connect to a Connect:Direct node.

To protect sensitive information that is stored in these files, use the fteObfuscate command. Specify the name of the file that is to be protected by using the **-f** flag. For example:

```
fteObfuscate -f <File to protect>
```

By default, the **fteObfuscate** command protects credentials with the default initial key. To protect credentials with a specific initial key, use the **-sf** parameter to specify the path to the file that contains the initial key. For example:

```
fteObfuscate -f <File to protect> -sf <initial key file>
```

CAUTION: The default initial key is the same for all IBM MQ installations. To protect passwords securely, supply an initial key that is unique to your installation when you encrypt passwords.

Attention:

1. Verify that all the sensitive information is protected after you run **fteObfuscate**.
2. Supply the protected file as normal to MFT.

If you specify an initial key file when you run the **fteObfuscate** command to protect credentials in the MFT configuration, you must also specify the same initial key file when MFT starts. The following mechanisms can be used to specify the name of the initial key file, in order of priority:

1. The **com.ibm.wmqfte.cred.keyfile** Java system property.

   Note: Before IBM MQ 9.3.1 and IBM MQ 9.3.0 Fix Pack 10, the name of this Java system property was misspelled as **com.ibm.wqmfte.cred.keyfile**. From IBM MQ 9.3.1 and IBM MQ 9.3.0 Fix Pack 10, Managed File Transfer uses both versions of the Java system property to maintain compatibility with earlier versions. If both Java system properties are set, the value of the correctly spelled property **com.ibm.wmqfte.cred.keyfile** is used.
2. Properties in the agent, logger, commands, and coordination property files.
3. The **commonCredentialsKeyFile** property in the `installation.properties` file.

For more information, see "Encrypting stored credentials in MFT" on page 560.

By default, the **fteObfuscate** command protects credentials with a protection system that is not compatible with MFT versions earlier than IBM MQ 9.2. To protect configuration files with the credentials protection system that is compatible with versions earlier than IBM MQ 9.2, specify the **-sp 0** parameter when the **fteObfuscate** command is run.

IBM MQ Internet Pass-Thru

The IBM MQ Internet Pass-Thru (MQIPT) configuration file can contain passwords that are used to access various resources.

Protect passwords in the MQIPT configuration file by using the **mqiptPW** command.

The **mqiptPW** command prompts for the password to be encrypted to be entered, and returns the encrypted password. Copy the encrypted password into the MQIPT configuration file.

By default, the **mqiptPW** command encrypts a password with the default initial key. To encrypt the password with a specific initial key, use the **-sf** parameter to specify the path to the file that contains the initial key.

CAUTION: The default initial key is the same for all IBM MQ installations. To protect passwords securely, supply an initial key that is unique to your installation when you encrypt passwords.

For more information, see Specifying the password encryption key.

If you specify an initial key file when you encrypt the key repository password, you must also specify the same initial key file when MQIPT starts. The following mechanisms can be used to specify the name of the initial key file, in order of priority:

1. The **-sf** parameter on the command that is used to start MQIPT.
2. The **MQS_MQIPTCRED_KEYFILE** environment variable.
3. The **com.ibm.mq.ipt.cred.keyfile** Java property.
4. A file named `mqipt_cred.key` in the MQIPT home directory. The MQIPT home directory is the directory that contains the MQIPT configuration file.

By default, the **mqiptPW** command protects credentials with a protection system that is not compatible with MQIPT versions earlier than IBM MQ 9.2. To protect passwords with the credentials protection system that is compatible with versions earlier than IBM MQ 9.2, use the **mqiptPW** command syntax that is supported in versions earlier than IBM MQ 9.2.

IBM MQ clients that use cryptographic hardware

You can configure IBM MQ clients to use PKCS #11 cryptographic hardware to store private keys and certificates that are used in TLS communications. To access PKCS #11 devices, you must provide a password as part of the configuration string that is supplied to the IBM MQ client.

Important: Passwords supplied by using the **CryptoHardware** field in the MQSCO structure, or the queue manager **SSLCRYP** attribute cannot be protected by using this mechanism.

You can protect this password by using the **runp11cred** command, which can be found in the `bin` folder in the IBM MQ installation directory.

The **runp11cred** command prompts for the password to be encrypted to be entered, and returns the encrypted password. The encrypted password must be copied into the cryptographic hardware configuration string.

For example, if your cryptographic hardware configuration string is the following:

```
GSK_PKCS11=/usr/lib/pkcs11/PKCS11_API.so;tokenlabel;Passw0rd;SYMMETRIC_CIPHER_ON
```

When the **runp11cred** command prompts you to enter the password, enter `Passw0rd`. The command returns a string that is similar to the following:

```
<P11>!2!0TyDxrRaS6JUsjON9zfK6S4wEHmSNF0/ZsOdCaTD2dc=!MdpCoxGnFqPtZ1dTLQ58kg==
```

Replace the password in the cryptographic hardware configuration string with the string that is returned by the **runp11cred** command, to give the following string that contains the encrypted password:

```
GSK_PKCS11=/usr/lib/pkcs11/PKCS11_API.so;tokenlabel;<P11>!2!0TyDxrRaS6JUsjON9zfK6S4wEHm SNF0/
ZsOdCaTD2dc=!MdpCoxGnFqPtZ1dTLQ58kg==;SYMMETRIC_CIPHER_ON
```

When the IBM MQ client application runs, specify the cryptographic hardware configuration string that contains the encrypted password in one of the following methods:

- The **SSLCryptoHardware** attribute in the SSL stanza of the client configuration file.
- The **MQSSLCRYP** environment variable.

By default, the **runp11cred** command encrypts a password with a default initial key. To protect a password with your own initial key, specify the name of the file that contains the initial key by using one of the following mechanisms, in order of priority:

1. The **-sf** parameter to the **runp11cred** command.
2. The **MQS_SSLCRYP_KEYFILE** environment variable.

CAUTION: The default initial key is the same for all IBM MQ installations. To protect passwords securely, supply an initial key that is unique to your installation when you encrypt passwords.

If you specify an initial key file when you encrypt the key repository password, you must also specify the name of the file that contains the initial key when the IBM MQ client runs. Specify the initial key file name by using one of the following mechanisms, in order of priority:

1. The **MQS_SSLCRYP_KEYFILE** environment variable.
2. The **SSLCryptoHardwareKeyFile** attribute in the **SSL** stanza of the client configuration file.

IBM MQ queue manager

The IBM MQ queue manager stores passwords internally in several attributes. For example, the queue manager **KEYRPWD** attribute. The queue manager automatically encrypts the password before it is stored in files on disk.

The password to the queue manager TLS key repository can be protected by using either the IBM MQ password protection system, or a key repository stash file. For more information about these two methods, see "Encrypting key repository passwords on AIX, Linux, and Windows" on page 290.

When the queue manager encrypts a password, the default initial key is used unless you specify your own initial key. To use your own initial key, set the queue manager **INITKEY** attribute to a unique, strong key before you set any encrypted queue manager attributes.

CAUTION: The default initial key is the same for all IBM MQ installations. To protect passwords securely, supply an initial key that is unique to your installation when you encrypt passwords.

Warning: If the initial key is modified after you set the value of attributes that are encrypted, the encrypted attributes are not re-encrypted with the new initial key. Therefore, changing the initial key without resupplying the key repository passphrase results in IBM MQ being unable to decrypt the key repository passphrase, and being unable to access the key repository.

For more information, see INITKEY.

IBM MQ C client applications

The IBM MQ C client libraries require passwords to access certain secured resources. For example, a TLS key repository for applications that use TLS to connect to the queue manager.

The key repository password can be protected by using either the IBM MQ password protection system, or a key repository stash file. For more information about these two methods, see "Encrypting key repository passwords on AIX, Linux, and Windows" on page 290.

To protect passwords with the IBM MQ password protection system, use the **runmqicred** command. The command is located in the *MQ_INSTALLATION_PATH*/bin directory.

The **runmqicred** command prompts for the password to be encrypted to be entered, and returns the encrypted password. The encrypted password can be used by the client application instead of a plain text password.

For example, if you choose to supply a TLS key repository password by using the *MQKEYRPWD* environment variable and your TLS keystore password is Passw0rd. When you run **runmqicred**, enter Passw0rd when prompted. The command returns a string that is similar to the following:

```
<MQI>!2!G41RxBuiNfJ3u0eYTD31G1hrL5NvVZLAlgZCX3Tn6d8=!pUD0ErDfDi9+JFVa0usS7w==
```

Set this string as the value for the *MQKEYRPWD* environment variable:

```
export MQKEYRPWD="<MQI>!2!G41RxBuiNfJ3u0eYTD31G1hrL5NvVZLAlgZCX3Tn6d8=!pUD0ErDfDi9+JFVa0usS7w=="
set MQKEYRPWD="<MQI>!2!G41RxBuiNfJ3u0eYTD31G1hrL5NvVZLAlgZCX3Tn6d8=!pUD0ErDfDi9+JFVa0usS7w=="
```

By default, the **runmqicred** command encrypts a password with the default initial key. To protect a password with your own initial key, use one of the following mechanisms to specify the name of the file that contains the key, in order of priority:

1. The **-sf** parameter to the **runmqicred** command.
2. The *MQS_MQI_KEYFILE* environment variable.

CAUTION: The default initial key is the same for all IBM MQ installations. To protect passwords securely, supply an initial key that is unique to your installation when you encrypt passwords.

If you specify an initial key file when you encrypt the password, you must also make the initial key available to the client application when it runs.

For more information, see "Supplying the key repository password for an IBM MQ MQI client on AIX, Linux, and Windows" on page 296.

Native HA configurations

V 9.4.0

Native HA log replication traffic between instances can be encrypted by using TLS. The certificates that are used to secure the log replication traffic are stored in a key repository that is specified in the **NativeHALocalInstance** stanza of the qm.ini file.

The key repository password can be protected by using either the IBM MQ password protection system, or a key repository stash file. For more information about these two methods, see "Encrypting key repository passwords on AIX, Linux, and Windows" on page 290.

To protect the Native HA key repository password with the IBM MQ password protection system, use the **runmqicred** command.

The **runmqicred** command prompts for the password to be encrypted to be entered, and returns the encrypted password. The encrypted password should be used instead of a plain text password. Set the value of the **KeyRepositoryPassword** attribute in the **NativeHALocalInstance** stanza of the `qm.ini` file to the encrypted password that the command returns.

By default, the **runmqicred** command encrypts a password with the default initial key. To protect a password with your own initial key, use one of the following mechanisms to specify the name of the file that contains the key, in order of priority:

1. The **-sf** parameter to the **runmqicred** command.
2. The *MQS_MQI_KEYFILE* environment variable.

⚠ **CAUTION:** The default initial key is the same for all IBM MQ installations. To protect passwords securely, supply an initial key that is unique to your installation when you encrypt passwords.

If you specify an initial key file when you encrypt the key repository password, you must also specify the same initial key file by using the **InitialKeyFile** attribute in the **NativeHALocalInstance** stanza of the `qm.ini` file.

For more information, see NativeHALocalInstance stanza of the qm.ini file.

IBM MQ queue manager (AuthToken stanza in the `qm.ini` file)

Linux V 9.4.0 AIX

From IBM MQ 9.3.4, IBM MQ MQI clients that connect to IBM MQ queue managers that run on AIX or Linux systems, can use authentication tokens to authenticate with the queue manager. The queue manager must be configured to accept authentication tokens and be able to access the token issuer's public key certificate or the secret key that is used to sign the token. The key repository that contains the trusted issuer's public key certificates or secret keys is secured with a password.

The key repository password can be protected by using either the IBM MQ password protection system, or a key repository stash file. For more information about these two methods, see "Encrypting key repository passwords on AIX, Linux, and Windows" on page 290.

To protect the authentication token key repository password with the IBM MQ password protection system, use the **runqmcred** command to encrypt the password.

The **runqmcred** command prompts for the password to be encrypted to be entered, and returns the encrypted password. The encrypted password must be used instead of a plain text password. Copy the encrypted password into a file and include the path to the file in the **KeyStorePwdFile** attribute of the **AuthToken** stanza in the `qm.ini` file.

By default, the **runqmcred** command encrypts a password with the default initial key. To encrypt the password with a specific initial key, use the **-sf** parameter to specify the path to the file that contains the initial key.

⚠ **CAUTION:** The default initial key is the same for all IBM MQ installations. To protect passwords securely, supply an initial key that is unique to your installation when you encrypt passwords.

Important: If you supply an initial key when you encrypt the password, the same initial key must be specified in the queue manager **INITKEY** attribute so that the queue manager can decrypt the password. If the queue manager **INITKEY** attribute is already set, use the same initial key when you run the **runqmcred** command. For more information about the queue manager **INITKEY** attribute, see INITKEY.

For example, to encrypt the authentication token keystore passwords with the initial key in the file /home/initial.key, issue the following command:

```
runqmcred -sf /home/initial.key
```

For more information, see "Configuring a queue manager to accept authentication tokens using a local keystore" on page 325.

The limits to protection through password encryption

IBM MQ supports AES-128 encryption for passwords that are stored in various configuration files. When you use Advanced Encryption Standard (AES) encryption to protect passwords in the IBM MQ configuration, you need to understand the limits to the protection that it provides.

Encrypting a password in the IBM MQ configuration files does not mean that the password is secure or protected. It only prevents the password from being easily recovered by someone who can access the encrypted password, but does not know the encryption key. IBM MQ processes require access to both the encrypted password and the decryption key to obtain the clear text password for use. Both these items of data must be stored on the file system in a location that is accessible to IBM MQ. Anyone who encrypts a password that is placed in a configuration file also requires access to the encryption key. If an attacker has access to the same set of files as IBM MQ, applying AES encryption to the password therefore provides only a minimal level of protection.

Nonetheless, encrypting passwords at rest is important to consider as it prevents the accidental disclosure of passwords and enables the sharing of configuration files, if the decryption key is not also shared.

In addition to ensuring that the file that contains the decryption key is not shared, care must be taken to ensure that the file is protected from other users on the system. While IBM MQ configuration files can be accessible to all users, restrict the permissions on the file that contains the decryption key to the minimum necessary. The user IDs that IBM MQ processes run as must be granted access to read the file that contains the decryption key. However, it is not necessary to grant access to read the file to a group, or all users on the system.

Protection of database authentication details

If your are using user name and password authentication to connect to the database manager, you can store them in the MQ XA credentials store to avoid storing the password in plain text in the qm.ini file.

Update XAOpenString for the resource manager

To use the credentials store you must modify XAOpenString in the qm.ini file. The string is used to connect to the database manager. You specify replaceable fields to identify where the user name and password are substituted within the XAOpenString string.

- The +USER+ field is replaced with the user name value stored in the XACredentials store.
- The +PASSWORD+ field is replaced with the password value stored in the XACredentials store.

The following examples show how to modify an XAOpenString to use the credentials file to connect to the database.

Connecting to a Db2 database

```
XAResourceManager:
  Name=mydb2
  SwitchFile=db2swit
  XAOpenString=db=mydbname,uid=+USER+,pwd=+PASSWORD+,toc=t
  ThreadOfControl=THREAD
```

Connecting to an Oracle database

```
XAResourceManager:
  Name=myoracle
  SwitchFile=oraswit
  XAOpenString=Oracle_XA+Acc=P/+USER+/+PASSWORD++SesTm=35
            +LogDir=/tmp+threads=true
  ThreadOfControl=THREAD
```

Work with the credentials for the database to the MQ XA credentials store

After you update the `qm.ini` file with the replaceable credential strings, you must add the user name and password to the MQ credentials store by using the **`setmqxacred`** command. You can also use **`setmqxacred`** to modify existing credentials, delete credentials, or list credentials. The following examples give some typical use cases:

Adding credentials

The following command securely saves the user name and password for the queue manager QM1 for the resource mqdb2.

```
setmqxacred -m QM1 -x mydb2 -u user1 -p Password2
```

Updating credentials

To update the user name and password used to connect to a database, re-issue the **`setmqxacred`** command with the new user-name and password:

```
setmqxacred -m QM1 -x mydb2 -u user3 -p Password4
```

You must restart the queue manager for the changes to take effect.

Deleting credentials

The following command deletes the credentials:

```
setmqxacred -m QM1 -x mydb2 -d
```

Listing credentials

The following command lists credentials:

```
setmqxacred -m QM1 -l
```

Related reference

`setmqxacred`

Securing Managed File Transfer

Directly after installation and with no modification, Managed File Transfer has a level of security that might be suitable for test or evaluation purposes in a protected environment. However, in a production environment, you must consider appropriately controlling who can start file transfer operations, who can read and write the files being transferred, and how to protect the integrity of files.

Related tasks

Restricting group authorities for MFT-specific resources

Managing authorities for MFT-specific resources

"Using Advanced Message Security with Managed File Transfer" on page 623

This scenario explains how to configure Advanced Message Security to provide message privacy for data being sent through a Managed File Transfer.

Related reference

Authorities for MFT to access file systems

commandPath MFT property

Authority to publish MFT Agents log and status messages

Encrypting stored credentials in MFT

Managed File Transfer (MFT) requires several user IDs and credentials, which are stored in two XML files, and you can obfuscate these using the **`fteObfuscate`** command.

Credential files

`MQMFTCredentials.xml`
: This file contains the user Id and credentials for connecting to agents and coordination and command queue managers. The credentials to access key stores for secure connections to queue managers are also stored in the same file.

  See "MFT and IBM MQ connection authentication" on page 563 for details of the property values that define the location of the `MQMFTCredentials.xml` file.

`ProtocolBridgeCredentials.xml`
: This file contains the user Id and credentials for connecting to protocol servers.

Encrypting credentials using the `fteObfuscate` command

The **`fteObfuscate`** command accepts the following parameters:

- **`-f`** *credentials_file_name* (required)

  Note: Deprecated This parameter replaces the **`-credentialsFile`** parameter that is deprecated from IBM MQ 9.2.0.
- **`-sp`** *protection_mode*
- **`-sf`** *credentials_key_file*
- **`-o`** *output_file_name*

See **`fteObfuscate`** for details of the parameters.

If you do not specify the protection mode, or a credentials key file, the command uses the default protection mode, and uses the latest algorithm, but with a fixed key to encrypt the credentials.

If you specify a protection mode of `0`, and do not specify a credentials key file, the command works as in previous releases of the product. You receive a warning message on the console indicating usage of deprecated protection.

If you specify a protection mode of `0`, and specify a credentials key file, you receive an error output on the console indicating that it is not valid to specify key file when using protection mode `0`.

If you specify the protection mode of `1`, and do not specify a credentials key file, the command uses the latest algorithm, but with a fixed key to encrypt the credentials.

If you specify the protection mode of `1`, and specify a credentials key file, the command encrypts the credentials with the latest algorithm.

If you specify the protection mode of `1`, or do not specify the protection mode, and specify a credentials key file that does not exist, an error is output on the console indicating that the file does not exist.

If you specify the protection mode of `1`, or do not specify the protection mode, and specify a credentials key file that is not readable, an error is output on the console indicating that the file is not readable..

If you specify the protection mode of 2, and do not specify a credentials key file, the command uses protection mode 2 to encrypt credentials using the latest algorithm and a fixed key to encrypt.

If you specify the protection mode of 2, and specify a credentials key file, the command uses protection mode 2 to encrypt credentials using the latest algorithm and a user specified key to encrypt.

If you specify the protection mode of 2, or do not specify the protection mode, and specify a credentials key file that does not exist, an error is output on the console indicating that the file does not exist.

If you specify the protection mode of 2, or do not specify the protection mode, and specify a credentials key file that is not readable, an error is output on the console indicating that the file is not readable..

Decrypting credentials

You can specify the path to the initial key file in various places. In order to decrypt credentials that were encrypted using an initial key other than the default one, the name of the file containing the initial key needs to be provided to MFT in one of the following ways, in this order of precedence:

1. By using a Java system property, for example:

   ```
   -Dcom.ibm.wmqfte.cred.keyfile=/usr/hime/credkeyfile.key
   ```

   Note:

   - Before IBM MQ 9.3.1 and IBM MQ 9.3.0 Fix Pack 10, the name of this Java system property was misspelled in the product code as `com.ibm.wqmfte.cred.keyfile`. From IBM MQ 9.3.1 and IBM MQ 9.3.0 Fix Pack 10, the spelling of the property name is corrected to be `com.ibm.wmqfte.cred.keyfile`. Managed File Transfer uses both versions of the Java system property when checking if a user has specified a file containing the initial key that should be used for encrypting and decrypting credentials. This allows you to use the correct spelling of the property name, while maintaining backwards compatibility with the old misspelled name. Note that if both Java system properties are set, then the value of the correctly spelled property `com.ibm.wmqfte.cred.keyfile` is used.
   - Before IBM MQ 9.3.1 and IBM MQ 9.3.0 Fix Pack 10, use the property `com.ibm.wqmfte.cred.keyfile`.

   .

2. By setting a property in an agent, command, coordination, or logger properties file. The name of the properties file, and the property that needs to be set in it are shown in the following table:

Property file	Property name
agent.properties	agentCredentialsKeyFile
command.properties	commandCredentialsKeyFile
coordination.properties	coordinationCredentialsKeyFile
logger.properties	loggerCredentialsKeyFile

3. In the installation.properties file.

   Instead of adding properties in individual properties files, you can add the **`commonCredentialsKeyFile`** property to the existing common `installation.properties` file, so that agent, logger and commands can use the same property.

If you have defined the various **`CredentialsKeyFile`** properties in multiple locations:

- The path of the credentials key file being used for the agent and logger is logged to the `output0.log` file for that agent or logger.
- The path of the credentials key file being used for the commands, is displayed on the console.

The Java system property **com.ibm.wmqfte.cred.keyfile** overrides all others. If the system property is not set, the agent looks into the `agent.properties` file, followed by the `installation.properties` file for the initial key file.

If the initial key file is still not found, and you have set the protection mode on the **fteObfuscate** command to `1`, the agent logs an error message in the `output0.log` file.

If you have set the protection mode to `0` on the **fteObfuscate** command, a warning message is logged indicating the deprecation.

The logger and commands follow the same steps for locating the initial key file.

Protocol Bridge and Connect:Direct Bridge

Protocol Bridge uses a properties file, `ProtocolBridgeProperties.xml`, for connecting to FTP, SFTP, and FTPS servers. This properties file contains connection attributes required to connect to these servers.

A bridge agent restart is required if you modify the value of the **credentialsFile** or **credentialsKeyFile** attributes in the `ProtocolBridgeProperties.xml` file.

One of the attributes is **credentialsFile**, and the value contains the path to an XML file containing UID, or PWD, or Key required to connect to these servers. The default value for the attribute is *ProtocolBridgeCredentials.xml* and the file is in your home directory, just like the `MQMFTCredentials.xml` file.

```
<tns:credentialsFile path="$HOME/ProtocolBridgeCredentials.xml" />
```

Just like *MQMFTCredentails.xml*, you can encrypt *ProtocolBridgeCredentials.xml* with the **fteObfuscate** command. For decryption purpose, you can specify the required path to a credentials key file using the additional element **credentialsKeyFile** as shown in the following text. The path can contain environment variables.

```
<tns:credentialsKeyFile path="$HOME/CredKey.key"/>
```

Note: Specifying a value for the **agentCredentialsKeyFile** agent property, **commonCredentialsKeyFile** property in the `installation.properties`, or through the system property **com.ibm.wqmfte.cred.keyfile**, does not have any impact on the value specified for the **credentialsKeyFile** attribute.

Similarly, Connect:Direct Bridge uses *ConnectDirectNodeProperties.xml* to connect to the Connect:Direct server. The XML file contains required connection information, along with an attribute that defines path to the credentials XML file. This credentials XML file contains UID, or PWD, and additional information required to connect to the Connect:Direct server.

```
<tns:credentialsFile path="$HOME/ ConnectDirectCredentials.xml" />
```

Just like the *ProtocolBridgeCredentials.xml* file, you can encrypt *ConnectDirectCredentials.xml* with the **fteObfuscate** command. For decryption purpose, you can specify the required path to a credentials key file using the additional element **credentialsKeyFile** as shown in the following text. The path can contain environment variables.

```
<tns:credentialsKeyFile path="$HOME/CredKey.key"/>
```

Note: Specifying a value for the **agentCredentialsKeyFile** agent property, **commonCredentialsKeyFile** property in the `installation.properties`, or through the system property **com.ibm.wqmfte.cred.keyfile** does not have any impact on the value specified for the **credentialsKeyFile** attribute.

You can specify the **credentialsKeyFile** element, without specifying the **credentialsFile** element in the *ProtocolBridgeProperties.xml* file.

If you do not specify the **credentialsFile** element, the default credential file *ProtocolBridgeCredentials.xml* is used by the protocol bridge agent, and the value of the key file specified in the **credentialsKeyFile** attribute is used to decrypt the credential file.

Similarly, you can specify the **credentialsKeyFile** element, without specifying the **credentialsFile** element in the *ConnectDirectNodeProperties.xml* file.

If you do not specify the **credentialsFile** element, the default credential file *ConnectDirectCredentials.xml* is used by the Connect:Direct bridge, and the value of the key file specified in the **credentialsKeyFile** attribute is used to decrypt the credential file.

Using the key from the data set on z/OS

z/OS

On z/OS, you can specify **MQMFTCredentials** and provide the credentials key file using a PDSE. See "Configuring MQMFTCredentials.xml on z/OS" on page 565.

Related reference

Which MFT command connects to which queue manager

MFT credentials file format

fteObfuscate (encrypt sensitive data)

MFT and IBM MQ connection authentication

Connection authentication allows a queue manager to be configured to authenticate applications by using a provided user ID and password. If the associated queue manager has security enabled, and requires credential details (user ID and password), the connection authentication feature must be enabled before a successful connection to a queue manager can be made. Connection authentication can be run in compatibility mode or MQCSP authentication mode.

Methods of supplying credential details

Many Managed File Transfer commands support the following methods of supplying credential details:

Details supplied by command line arguments.
The credential details can be specified by using the **-mquserid** and **-mqpassword** parameters. If the **-mqpassword** is not supplied, then the user is asked for the password where the input is not displayed.

Details supplied from a credentials file: MQMFTCredentials.xml.
The credential details can be predefined in a `MQMFTCredentials.xml` file either as clear text or obfuscated text.

Multi For information about setting up an MQMFTCredentials.xml file on IBM MQ for Multiplatforms see "Configuring MQMFTCredentials.xml on Multiplatforms" on page 564.

z/OS For information about setting up an MQMFTCredentials.xml file on IBM MQ for z/OS see "Configuring MQMFTCredentials.xml on z/OS" on page 565.

Precedence

The precedence of determining the credential details is:

1. Command line argument.
2. `MQMFTCredentials.xml` index by associated queue manager and user running the command.
3. `MQMFTCredentials.xml` index by associated queue manager.
4. Default backward compatibility mode where no credential details are supplied to allow compatibility with previous releases of IBM MQor IBM WebSphere MQ

Notes:

- The **`fteStartAgent`** and **`fteStartLogger`** commands do not support the command line argument **`-mquserid`**, or **`-mqpassword`**, and the credential details can only be specified with the `MQMFTCredentials.xml` file.
- z/OS

  On z/OS, the password must be uppercase, even if the user's password has lowercase letters. For example, if the user's password was "password", it would have to be entered as "PASSWORD".

Related reference

Which MFT command connects to which queue manager

MFT credentials file format

Configuring MQMFTCredentials.xml on Multiplatforms

If Managed File Transfer (MFT) is configured with security enabled, connection authentication requires all MFT commands that connect with a queue manager to supply user ID and password credentials. Similarly, MFT loggers might be required to specify a user ID and password when connecting to a database. This credential information can be stored in the MFT credentials file.

About this task

The elements in the `MQMFTCredentials.xml` file must conform to the `MQMFTCredentials.xsd` schema. For information about the format of `MQMFTCredentials.xml`, see MFT credentials file format.

You can find a sample credentials file in the `MQ_INSTALLATION_PATH/mqft/samples/credentials` directory.

You can have one MFT credentials file for the coordination queue manager, one for the command queue manager, one for each agent, and one for each logger. Alternatively, you can have one file which is used by everything in your topology.

The default location of the MFT credentials file is as follows:

Linux AIX **AIX and Linux**

`$HOME`

Windows **Windows**

`%USERPROFILE%` or `%HOMEDRIVE%%HOMEPATH%`

If the credentials file is stored in a different location, then you can use the following properties to specify where the commands should look for it:

Table 97. : Properties that define the location of the MQMFTCredentials.xml file for various commands.

Type of command	Property file	Property name
Command which connects to the coordination queue manager	coordination.properties	coordinationQMgrAuthenticationCredentialsFile
Command which connects to the command queue manager	connection.properties	connectionQMgrAuthenticationCredentialsFile
Command that connects to an agent process	agent.properties	agentQMgrAuthenticationCredentialsFile
Command that connects to a logger process	logger.properties	loggerQMgrAuthenticationCredentialsFile

Table 98. : Properties that define the location of the MQMFTCredentials.xml file for agents and logger processes.

Type of command	Property file	Property name
MFT agents	agent.properties	agentQMgrAuthenticationCredentialsFile
MFT loggers	logger.properties	loggerQMgrAuthenticationCredentialsFile

For details about what commands and processes connect to which queue manager, see Which MFT commands and processes connect to which queue manager.

Instead of adding properties in individual properties files, you can add the **`commonCredentialsKeyFile`** property to the existing common `installation.properties` file, so that agent, logger and commands can use the same property.

Because the credentials file contains user ID and password information, it requires special permissions to prevent unauthorized access to it:

Linux AIX **AIX and Linux**

```
chown <agent owner userid>
chmod 600
```

Windows **Windows**

Ensure that inheritance is not enabled, and then remove all of the user IDs except those running the agent or logger that will be using the credentials file.

The credential details used to connect to an MFT coordination queue manager, in the IBM MQ Explorer Managed File Transfer plug-in depends on the type of configuration:

Global (configuration on local disk)
A global configuration uses the credentials file specified in the coordination and command properties.

Local (defined within IBM MQ Explorer):
A local configuration uses the properties of the connection details of the associated queue manager in IBM MQ Explorer.

Related tasks

"Enabling connection authentication for MFT" on page 567
Connection authentication of the IBM MQ Explorer MFT Plugin connecting with a coordination queue manager or command queue manager, and connection authentication for a Managed File Transfer agent connecting with a coordination queue manager or command queue manager can be run in compatibility mode or MQCSP authentication mode.

Creating an IBM MQ File Transfer structure

Related reference

MFT credentials file format

Encrypting stored credentials in MFT

`fteObfuscate`: encrypt sensitive data

z/OS Configuring `MQMFTCredentials.xml` on z/OS

If Managed File Transfer (MFT) is configured with security enabled, connection authentication requires all MFT agents, and commands that connect to a queue manager, to supply user ID and password credentials.

Similarly, MFT loggers might be required to specify a user ID and password when connecting to a database.

This credential information can be stored in the MFT credentials file. Note that the credentials files are optional, however, it is easier to define the file or files that you require before you customize the environment.

In addition to this, if you have credentials files, you receive fewer warning messages. The warning messages inform you that MFT considers that queue manager security is off, and therefore you are not supplying authentication details.

You can find a sample credentials file in the `MQ_INSTALLATION_PATH/mqft/samples/credentials` directory.

Here is an example of an `MQMFTCredentials.xml`file:

```
<?xml version="1.0" encoding="IBM-1047"?>
<tns:mqmftCredentials xmlns:tns="http://wmqfte.ibm.com/MFTCredentials"
xmlns:xsi="https://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://wmqfte.ibm.com/MFTCredentials MFTCredentials.xsd">
  <tns:qmgr name="MQPH" user="ADMIN" mqUserId="JOHNDOEH" mqPassword="cXXXX" />
  <tns:qmgr name="MQPI" user="ADMIN" mqUserId="JOHNDOEI" mqPassword="yXXXX" />
  <tns:qmgr name="MQPH" mqUserId="NONEH" mqPassword="yXXXX" />
  <tns:qmgr name="MQPI" mqUserId="NONEI" mqPassword="yXXXX" />
</tns:mqmftCredentials>
```

When a job with userid ADMIN needs to connect to queue manager MQPH, it passes user ID *JOHNDOEH* and uses password *cXXXX*.

If the job is run by any other user ID, and connects MQPH, that job passes user ID *NONEH* and password *yXXXX*.

The default location for the `MQMFTCredentials.xml` file is the user's home directory on z/OS UNIX System Services (USS). It is also possible to store the file in either a different location on USS, or in a member within a partitioned data set.

If the credentials file is stored in a different location, then you can use the following properties to specify where the commands should look for it:

Table 99. : Properties that define the location of the MQMFTCredentials.xml file for various commands.

Type of command	Property file	Property name
Command which connects to the coordination queue manager	coordination.properties	coordinationQMgrAuthenticationCredentialsFile
Command which connects to the command queue manager	connection.properties	connectionQMgrAuthenticationCredentialsFile
Command that connects to an agent process	agent.properties	agentQMgrAuthenticationCredentialsFile
Command that connects to a logger process	logger.properties	loggerQMgrAuthenticationCredentialsFile

Table 100. : Properties that define the location of the MQMFTCredentials.xml file for agents and logger processes.

Type of command	Property file	Property name
MFT agents	agent.properties	agentQMgrAuthenticationCredentialsFile
MFT loggers	logger.properties	loggerQMgrAuthenticationCredentialsFile

For details about what commands and processes connect to which queue manager, see Which MFT commands and processes connect to which queue manager.

To create the credentials file within a partitioned data set, carry out the following steps:

- Create a PDSE with format VB and logical record length (Lrecl) 200.
- Create a member within the data set, make a note of the data set and member, and add the following code to the member:

```
<?xml version="1.0" encoding="IBM-1047"?>
<tns:mqmftCredentials xmlns:tns="http://wmqfte.ibm.com/MQMFTCredentials"
xmlns:xsi="https://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://wmqfte.ibm.com/MFTCredentials MQMFTCredentials.xsd">
  <!—-credentials information goes here-->
</tns:mqmftCredentials>
```

You can protect the credentials file using a security product, for example, RACF, but the user IDs running the Managed File Transfer commands, and administering the agent and logger processes, need read access to this file.

You can obscure information in this file using the JCL in member BFGCROBS. This takes the file and encrypts the IBM MQ user ID and password. For example member BFGCROBS takes the line

```
<tns:qmgr name="MQPI" user="JOHNDOE2" mqUserId="JOHNDOE1" mqPassword="yXXXX" />
```

and creates

```
<tns:qmgr mqPasswordCipher="e977c61e9b9c363c" mqUserIdCipher="c394c5887867157c"
name="MQPI" user="JOHNDOE2"/>
```

If you want to keep the user ID to IBM MQ user ID mapping, you can add comments to the file. For example

```
<!--      name="MQPI" user="ADMIN"    mqUserId="JOHNDOE1 -->
```

These comments are unchanged by the obscuring process.

Note that the content is obscured, not strongly encrypted. You should limit which user IDs have access to the file.

Related tasks

“Configuring MQMFTCredentials.xml on Multiplatforms” on page 564
If Managed File Transfer (MFT) is configured with security enabled, connection authentication requires all MFT commands that connect with a queue manager to supply user ID and password credentials. Similarly, MFT loggers might be required to specify a user ID and password when connecting to a database. This credential information can be stored in the MFT credentials file.

Enabling connection authentication for MFT

Connection authentication of the IBM MQ Explorer MFT Plugin connecting with a coordination queue manager or command queue manager, and connection authentication for a Managed File Transfer agent connecting with a coordination queue manager or command queue manager can be run in compatibility mode or MQCSP authentication mode.

About this task

MQCSP authentication mode is the default.

For connection authentication for the IBM MQ Explorer Managed File Transfer plugin or for Managed File Transfer agents that connect to a queue manager using the CLIENT transport, passwords longer than 12 characters are only supported for MQCSP authentication mode. If you specify a password greater than 12 characters in length when authorizing using compatibility mode, then an error occurs and the agent does not authenticate with the queue manager. See the BFGAG0187E message in Diagnostic messages: BFGAG0001 - BFGAG9999.

Procedure

- To select the connection authentication mode for a coordination queue manager or command queue manager in IBM MQ Explorer, complete the following steps:
  a) Select the queue manager that you want to connect to.
  b) Right click, and select **Connection Details->Properties** from the pop-up menu.
  c) Click the **Userid** tab.
  d) Make sure that the check box for the mode of connection authentication that you want to use is selected:
     - By default, the **User identification compatibility mode** check box is unselected. This means that if the **Enable user identification** check box is selected, the IBM MQ Explorer will use MQCSP authentication when connecting to the queue manager. If IBM MQ Explorer needs to connect to the queue manager using compatibility mode instead of MQCSP authentication, ensure that both the **Enable user identification** and the **User identification compatibility mode** check boxes are selected.
- To enable or disable MQCSP authentication mode for a Managed File Transfer agent by using the `MQMFTCredentials.xml` file, add the parameter **`useMQCSPAuthentication`** to the `MQMFTCredentials.xml` file for the relevant user.

  The **`useMQCSPAuthentication`** parameter has the following values:

  true
  MQCSP authentication mode is used to authenticate the user with the queue manager.

  `true` is the default value. If the **`useMQCSPAuthentication`** parameter is not specified, it is by default set to `true` and MQCSP authentication mode is used to authenticate the user with the queue manager..

  false
  Compatibility mode is used to authenticate the user with the queue manager.

  The following example shows how to set the **`useMQCSPAuthentication`** parameter in the `MQMFTCredentials.xml` file:

```
<tns:qmgr name="CoordQueueMgr" user="ernest" mqUserId="ernest"
    mqPassword="AveryL0ngPassw0rd2135" useMQCSPAuthentication="true"/>
```

Related concepts

"MQCSP password protection" on page 31
Authentication credentials that are specified in the MQCSP structure can be either protected by using the IBM MQ MQCSP password protection feature, or encrypted by using TLS encryption.

Related reference

"MFT and IBM MQ connection authentication" on page 563
Connection authentication allows a queue manager to be configured to authenticate applications by using a provided user ID and password. If the associated queue manager has security enabled, and requires credential details (user ID and password), the connection authentication feature must be enabled before a successful connection to a queue manager can be made. Connection authentication can be run in compatibility mode or MQCSP authentication mode.

MFT credentials file format

MFT sandboxes

You can restrict the area of the file system that the agent can access as part of a transfer. The area that the agent is restricted to is called the sandbox. You can apply restrictions to either the agent or to the user that requests a transfer.

Sandboxes are not supported when the agent is a protocol bridge agent or a Connect:Direct bridge agent. You can not use agent sandboxing for agents that need to transfer to or from IBM MQ queues.

Related reference

"Working with MFT agent sandboxes" on page 569
To add an additional level of security to Managed File Transfer, you can restrict the area of a file system that an agent can access.

"Working with MFT user sandboxes" on page 570
You can restrict the area of the file system that files can be transferred into and out of based on the MQMD user name that requests the transfer.

Working with MFT agent sandboxes

To add an additional level of security to Managed File Transfer, you can restrict the area of a file system that an agent can access.

You cannot use agent sandboxing for agents that transfer to or from IBM MQ queues. Restricting access to IBM MQ queues with sandboxing can be implemented instead by using user sandboxing which is the recommended solution for any sandboxing requirements. For more information about user sandboxing, see "Working with MFT user sandboxes" on page 570

To enable agent sandboxing, add the following property to the `agent.properties` file for the agent you want to restrict:

```
sandboxRoot=[!]restricted_directory_nameseparator...separator[!]restricted_directory_name
```

where:

- *restricted_directory_name* is a directory path to be allowed or denied.
- `!` is optional and specifies that the following value for *restricted_directory_name* is denied (excluded). If `!` is not specified *restricted_directory_name* is an allowed (included) path.
- *separator* is the platform-specific separator.

For example, if you want to restrict the access that AGENT1 has to the `/tmp` directory only, but not allow the subdirectory `private` to be accessed, set the property as follows in the `agent.properties` file belonging to AGENT1: `sandboxRoot=/tmp:!/tmp/private`.

The sandboxRoot property is described in Advanced agent properties.

Both agent and user sandboxing are not supported on protocol bridge agents or on Connect:Direct bridge agents.

Working in a sandbox on AIX, Linux, and Windows platforms

ALW On AIX, Linux, and Windows platforms, sandboxing restricts which directories a Managed File Transfer Agent can read from and write to. When sandboxing is activated, the Managed File Transfer Agent can read and write to the directories specified as allowed, and any subdirectories that the specified directories contain unless the subdirectories are specified as denied in the sandboxRoot. Managed File Transfer sandboxing does not take precedence over operating system security. The user that started the Managed File Transfer Agent must have the appropriate operating system level access to any directory to be able to read from or write to the directory. A symbolic link to a directory is not followed if the directory linked to is outside the specified sandboxRoot directories (and subdirectories).

Working in a sandbox on z/OS

z/OS On z/OS, sandboxing restricts the data set name qualifiers that the Managed File Transfer Agent can read from and write to. The user that started the Managed File Transfer Agent must have the correct operating system authorities to any data sets involved. If you enclose a sandboxRoot data set name qualifier value in double quotation marks, the value follows the normal z/OS convention and is treated as fully qualified. If you omit the double quotation marks, the sandboxRoot is prefixed with the current user ID. For example, if you set the sandboxRoot property to the following: `sandboxRoot=//`

`test`, the agent can access the following data sets (in standard z/OS notation) `//`*`username`*`.test.**` At run time, if the initial levels of the fully resolved data set name do not match the sandboxRoot, the transfer request is rejected.

Working in a sandbox on IBM i systems

IBM i For files in the integrated file system on IBM i systems, sandboxing restricts which directories a Managed File Transfer Agent can read from and write to. When sandboxing is activated, the Managed File Transfer Agent can read and write to the directories specified as allowed, and any subdirectories that the specified directories contain unless the subdirectories are specified as denied in the sandboxRoot. Managed File Transfer sandboxing does not take precedence over operating system security. The user that started the Managed File Transfer Agent must have the appropriate operating system level access to any directory to be able to read from or write to the directory. A symbolic link to a directory is not followed if the directory linked to is outside the specified sandboxRoot directories (and subdirectories).

Related reference

"Additional checks for wildcard transfers" on page 573
If an agent has been configured with a user or agent sandbox in order to restrict the locations that the agent can transfer files to and from, you can specify that additional checks are to be made on wildcard transfers for that agent.

"Working with MFT agent sandboxes" on page 569
To add an additional level of security to Managed File Transfer, you can restrict the area of a file system that an agent can access.

The MFT `agent.properties` file

Working with MFT user sandboxes

You can restrict the area of the file system that files can be transferred into and out of based on the MQMD user name that requests the transfer.

User sandboxes are not supported when the agent is a protocol bridge agent or a Connect:Direct bridge agent.

To enable user sandboxing, add the following property to the `agent.properties` file for the agent that you want to restrict:

```
userSandboxes=true
```

When this property is present and set to true the agent uses the information in the *`MQ_DATA_PATH`*`/mqft/config/`*`coordination_qmgr_name`*`/agents/`*`agent_name`*`/UserSandboxes.xml` file to determine which parts of the file system the user who requests the transfer can access.

The `UserSandboxes.xml` XML is composed of an `<agent>` element that contains zero or more `<sandbox>` elements. These elements describe which rules are applied to which users. The `user` attribute of the `<sandbox>` element is a pattern that is used to match against the MQMD user of the request.

The file `UserSandboxes.xml` is periodically reloaded by the agent and any valid changes to the file will affect the behavior of the agent. The default reload interval is 30 seconds. This interval can be changed by specifying the agent property xmlConfigReloadInterval in the `agent.properties` file.

If you specify the `userPattern="regex"` attribute or value, the `user` attribute is interpreted as a Java regular expression. For more information, see Regular expressions used by MFT.

If you do not specify the `userPattern="regex"` attribute or value the `user` attribute is interpreted as a pattern with the following wildcard characters:

- asterisk (*), which represents zero or more characters
- question mark (?), which represents exactly one character

Matches are performed in the order that the <sandbox> elements are listed in the file. Only the first match is used, all following potential matches in the file are ignored. If none of the <sandbox> elements specified in the file match the MQMD user associated with the transfer request message, the transfer cannot access the file system. When a match has been found between the MQMD user name and a `user` attribute, the match identifies a set of rules inside a <sandbox> element that are applied to the transfer. This set of rules is used to determine which files, or data sets, can be read from or written to as part of the transfer.

Each set of rules can specify a <read> element, which identifies which files can be read, and a <write> element which identifies which files can be written. If you omit the <read> or <write> elements from a set of rules, it is assumed that the user associated with that set of rules is not allowed to perform any reads or any writes, as appropriate.

Note: The <read> element must be before the <write> element, and the <include> element must be before the <exclude> element, in the `UserSandboxes.xml` file.

Each <read> or <write> element contains one or more patterns that are used to determine whether a file is in the sandbox and can be transferred. Specify these patterns by using the <include> and <exclude> elements. The name attribute of the <include> or <exclude> element specifies the pattern to be matched. An optional `type` attribute specifies whether the name value is a file or queue pattern. If the `type` attribute is not specified, the agent treats the pattern as a file or directory path pattern. For example:

```
<tns:read>
    <tns:include name="/home/user/**"/>
    <tns:include name="USER.**" type="queue"/>
    <tns:exclude name="/home/user/private/**"/>
</tns:read>
```

The <include> and <exclude> name patterns are used by the agent to determine whether files, data sets, or queues can be read from or written to. An operation is allowed if the canonical file path, data set, or queue name matches at least one of the included patterns and exactly zero of the excluded patterns. The patterns specified by using the name attribute of the <include> and <exclude> elements use the path separators and conventions appropriate to the platform that the agent is running on. If you specify relative file paths, the paths are resolved relative to the `transferRoot` property of the agent.

When specifying a queue restriction, a syntax of QUEUE@QUEUEMANAGER is supported, with the following rules:

- If the at character (@) is missing from the entry, the pattern is treated as a queue name that can be accessed on any queue manager. For example, if the pattern is `name` it is treated the same way as `name@**`.
- If the at character (@) is the first character in the entry, the pattern is treated as a queue manager name and all queues on the queue manager can be accessed. For example, if the pattern is `@name` it is treated the same way as `**@name`..

The following wildcard characters have special meaning when you specify them as part of the `name` attribute of the <include> and <exclude> elements:

*****
A single asterisk matches zero or more characters in a directory name, or in a qualifier of a data set name or queue name.

?
A question mark matches exactly one character in a directory name, or in a qualifier of a data set name or queue name.

******
Two asterisk characters match zero or more directory names, or zero or more qualifiers in a data set name or queue name. Also, paths that end with a path separator have an implicit "**" added to the end of the path. So `/home/user/` is the same as `/home/user/**`.

For example:

- `/**/test/**` matches any file that has a `test` directory in its path
- `/test/file?` matches any file inside the `/test` directory that starts with the string `file` followed by any single character
- `c:\test\*.txt` matches any file inside the `c:\test` directory with a `.txt` extension
- `c:\test\**\*.txt` matches any file inside the `'c:\test` directory, or one of its subdirectories that has a `.txt` extension
- z/OS `//'TEST.*.DATA'` matches any data set that has the first qualifier of TEST, has any second qualifier, and a third qualifier of DATA.
- *@QM1 matches any queue on the queue manager QM1 that has a single qualifier.
- `TEST.*.QUEUE@QM1` matches any queue on the queue manager QM1 that has the first qualifier of TEST, has any second qualifier, and a third qualifier of QUEUE.
- **@QM1 matches any queue on the queue manager QM1.

Symbolic links

You must fully resolve any symbolic links that you use in file paths in the `UserSandboxes.xml` file by specifying hard links in the `<include>` and `<exclude>` elements. For example, if you have a symbolic link where `/var` maps to `/SYSTEM/var`, you must specify this path as `<tns:include name="/SYSTEM/var"/>`, otherwise the intended transfer fails with a user sandbox security error.

Example

This example shows how to allow the user with the MQMD user name `guest` to transfer any file from the `/home/user/public` directory or any of its subdirectories on the system where the agent AGENT_JUPITER is running, by adding the following `<sandbox>` element to the file `UserSandboxes.xml` in AGENT_JUPITER's configuration directory:

```
<?xml version="1.0" encoding="UTF-8"?>
<tns:userSandboxes
        xmlns:tns="http://wmqfte.ibm.com/UserSandboxes"
        xmlns:xsi="https://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://wmqfte.ibm.com/UserSandboxes UserSandboxes.xsd">
    <tns:agent>
        <tns:sandbox user="guest">
            <tns:read>
                <tns:include name="/home/user/public/**"/>
             </tns:read>
        </tns:sandbox>
    </tns:agent>
</tns:userSandboxes>
```

Example

This example shows how to allow any user with the MQMD user name `account` followed by a single digit, for example `account4`, to complete the following actions:

- Transfer any file from the `/home/account` directory or any of its subdirectories, excluding the `/home/account/private` directory on the system where the agent AGENT_SATURN is running
- Transfer any file to the `/home/account/output` directory or any of its subdirectories on the system where the agent AGENT_SATURN is running
- Read messages from queues on the local queue manager starting with the prefix `ACCOUNT.` unless it starts with `ACCOUNT.PRIVATE.` (that is has `PRIVATE` at the second level).
- Transfer data onto queues starting with the prefix `ACCOUNT.OUTPUT.` on any queue manager.

To allow a user with the MQMD user name `account` to complete these actions, add the following `<sandbox>` element to the file `UserSandboxes.xml`, in AGENT_SATURN's configuration directory:

```
<?xml version="1.0" encoding="UTF-8"?>
<tns:userSandboxes
            xmlns:tns="http://wmqfte.ibm.com/UserSandboxes"
            xmlns:xsi="https://www.w3.org/2001/XMLSchema-instance"
            xsi:schemaLocation="http://wmqfte.ibm.com/UserSandboxes UserSandboxes.xsd">
    <tns:agent>
        <tns:sandbox user="account[0-9]" userPattern="regex">
            <tns:read>
                <tns:include name="/home/account/**"/>
                <tns:include name="ACCOUNT.**" type="queue"/>
                <tns:exclude name="ACCOUNT.PRIVATE.**" type="queue"/>
                <tns:exclude name="/home/account/private/**"/>
                                                </tns:read>
            <tns:write>
                <tns:include name="/home/account/output/**"/>
                <tns:include name="ACCOUNT.OUTPUT.**" type="queue"/>
            </tns:write>
        </tns:sandbox>
    </tns:agent>
</tns:userSandboxes>
```

Related reference

"Additional checks for wildcard transfers" on page 573

If an agent has been configured with a user or agent sandbox in order to restrict the locations that the agent can transfer files to and from, you can specify that additional checks are to be made on wildcard transfers for that agent.

The MFT `agent.properties` file

Additional checks for wildcard transfers

If an agent has been configured with a user or agent sandbox in order to restrict the locations that the agent can transfer files to and from, you can specify that additional checks are to be made on wildcard transfers for that agent.

additionalWildcardSandboxChecking property

To enable additional checking for wildcard transfers, add the following property to the `agent.properties` file for the agent that you want to check.

```
additionalWildcardSandboxChecking=true
```

When this property is set to true, and the agent makes a transfer request that attempts to read a location that is outside the defined sandbox for file matching of the wildcard, the transfer fails. If there are multiple transfers within one transfer request, and one of these requests fails due to it attempting to read a location outside of the sandbox, the entire transfer fails. If checking fails, the reason for failure is given in an error message.

If the additionalWildcardSandboxChecking property is omitted from an agent's `agent.properties` file or is set to false, no additional checks are made on wildcard transfers for that agent.

Error messages for wildcard checking

The messages that are reported when a wildcard transfer request is made to a location outside a configured sandbox location are as follows.

The following message occurs when a wildcard file path in a transfer request is located outside of the restricted sandbox:

```
BFGSS0077E: Attempt to read file path: path has been denied.
The file path is located outside of the restricted transfer sandbox.
```

The following message occurs when a transfer within a multiple transfer request contains a wildcard transfer request where the path is located outside of the restricted sandbox:

```
BFGSS0078E: Attempt to read file path: path has been ignored as another transfer
item in the managed transfer attempted to read outside of the restricted transfer sandbox.
```

The following message occurs when a file is located outside of the restricted sandbox:

```
BFGSS0079E: Attempt to read file file path has been denied.
The file is located outside of the restricted transfer sandbox.
```

The following message occurs in a multiple transfer request where another wildcard transfer request has caused this one to be ignored:

```
BFGSS0080E: Attempt to read file: file path has been ignored as another transfer
item in the managed transfer attempted to read outside of the restricted transfer sandbox.
```

In the case of single file transfers that do not include wildcards, the message that is reported when the transfer involves a file that is located outside of the sandbox is unchanged from earlier releases:

```
Fails with BFGIO0056E: Attempt to read file "FILE" has been denied.
The file is located outside of the restricted transfer sandbox.
```

Related reference

"Working with MFT user sandboxes" on page 570
You can restrict the area of the file system that files can be transferred into and out of based on the MQMD user name that requests the transfer.

"Working with MFT agent sandboxes" on page 569
To add an additional level of security to Managed File Transfer, you can restrict the area of a file system that an agent can access.

The MFT `agent.properties` file

Configuring SSL or TLS encryption for MFT

You can use SSL or TLS can be used with IBM MQ Managed File Transfer to secure the communication between agents and their agent queue managers, commands and the queue managers that they are connecting to, and the various queue manager to queue manager connections within your topology.

Before you begin

You can use SSL or TLS encryption to encrypt messages that are flowing through an IBM MQ Managed File Transfer topology. These include:

- Messages that pass between an agent and its agent queue manager.
- Messages for commands and the queue managers that they are connecting to.
- Internal messages that flow between the agent queue managers, command queue managers and coordination queue manager within the topology.

About this task

For general information about using SSL with IBM MQ, see "Working with SSL/TLS" on page 270. In IBM MQ terms, Managed File Transfer is a standard Java client application.

Follow these steps to use SSL with Managed File Transfer:

Procedure

1. Create a truststore file and optionally a keystore file (these files can be the same file). If you do not need client-authentication (that is, SSLCAUTH=OPTIONAL on channels) you do not need to provide a keystore. You require a truststore only to authenticate the queue manager's certificate.

   The key algorithm used for creating certificates for the truststore and keystores must be RSA to work with IBM MQ.
2. Set up your IBM MQ queue manager to use SSL.

For information about setting up a queue manager to use SSL using IBM MQ Explorer for example, see Configuring SSL on queue managers.

3. Save the truststore file and keystore file (if you have one) in a suitable location. A suggested location is the *config_directory*/*coordination_qmgr*/agents/*agent_name* directory.
4. Set the SSL properties as required for each SSL-enabled queue manager in the appropriate Managed File Transfer properties file. Each set of properties refers to a separate queue manager (agent, coordination, and command), although one queue manager might perform two or more of these roles.

   One of the **CipherSpec** or **CipherSuite** properties is required, otherwise the client tries to connect without SSL. Both the **CipherSpec** or **CipherSuite** properties are provided because of the terminology differences between IBM MQ and Java. Managed File Transfer accepts either property and does the necessary conversion, so you do not need to set both properties. If you do specify both the **CipherSpec** or **CipherSuite** properties, **CipherSpec** takes precedence.

   The **PeerName** property is optional. You can set the property to the Distinguished Name of the queue manager that you want to connect to. Managed File Transfer rejects connections to an incorrect SSL server with a Distinguished Name that does not match.

   Set the **SslTrustStore** and **SslKeyStore** properties to file names that point to the truststore and keystore files. If you are setting up these properties for an agent that is already running, stop and restart the agent to reconnect in SSL mode.

   Properties files contain plain-text passwords so consider setting appropriate file system permissions.

   For more information about SSL properties, see "SSL/TLS properties for MFT" on page 575.
5. If an agent queue manager uses SSL, you cannot provide the necessary details when you create the agent. Use the following steps to create the agent:
   a) Create the agent by using the **fteCreateAgent** command. You receive a warning about being unable to publish the existence of the agent to the coordination queue manager.
   b) Edit the `agent.properties` file that was created by the previous step to add the SSL information. When the agent is successfully started, the publish is attempted again.
6. If agents or instances of the IBM MQ Explorer are running while the SSL properties in the `agent.properties` file or `coordination.properties` file are changed, you must restart the agent or IBM MQ Explorer.

Related reference

The MFT `agent.properties` file

SSL/TLS properties for MFT

Some MFT properties files include SSL and TLS properties. You can use SSL or TLS with IBM MQ and Managed File Transfer to prevent unauthorized connections between agents and queue managers, and to encrypt message traffic between agents and queue managers.

The following MFT properties files include SSL properties:

- SSL/TLS properties for the MFT `agent.properties` file
- SSL/TLS properties for the MFT `coordination.properties` file
- SSL/TLS properties for the MFT `command.properties` file
- SSL/TLS properties for the MFT `logger.properties` file

For information about using SSL or TLS with Managed File Transfer, see "Configuring SSL or TLS encryption for MFT" on page 574.

From IBM WebSphere MQ 7.5, you can use environment variables in some Managed File Transfer properties that represent file or directory locations. This allows the locations of files or directories that are used when running parts of the product to vary depending on environment changes, such as which user is running the process. For more information, see The use of environment variables in MFT properties.

Related concepts

MFT configuration options on Multiplatforms

Related reference

The use of environment variables in MFT properties

Connecting to a queue manager in client mode with channel authentication

IBM MQ uses channel authentication records to control more precisely access at a channel level. This means that by default newly created queue managers reject client connections from the Managed File Transfer component.

For more information about channel authentication, see "Channel authentication records" on page 51.

If the channel authentication configuration for the SVRCONN used by Managed File Transfer specifies a non-privileged MCAUSER ID, you must grant specific authority records for the queue manager, queues, and topics, to allow the Managed File Transfer Agent and commands to work correctly. Use the MQSC command SET CHLAUTH or the PCF command Set Channel Authentication Record to create, modify, or remove channel authentication records. For all Managed File Transfer agents that you want to connect to the IBM MQ queue manager, you can either set up an MCAUSER ID to use for all your agents, or set up a separate MCAUSER ID for each agent.

Grant each MCAUSER ID the following permissions:

- Authority records required for the queue manager:
  - connect
  - setid
  - inq
- Authority records required for queues.

  For all agent-specific queues, that is queue names that end in *agent_name* in the following list, you must create these queue authority records for each agent that you want to connect to the IBM MQ queue manager by using a client connection.

  - put, get, dsp (SYSTEM.DEFAULT.MODEL.QUEUE)
  - put, get, setid, browse (SYSTEM.FTE.COMMAND.*agent_name*)
  - put, get (SYSTEM.FTE.DATA.*agent_name*)
  - put, get (SYSTEM.FTE.REPLY.*agent_name*)
  - put, get, inq, browse (SYSTEM.FTE.STATE.*agent_name*)
  - put, get, browse (SYSTEM.FTE.EVENT.*agent_name*)
  - put, get (SYSTEM.FTE)
- Authority records required for topics:
  - sub, pub (SYSTEM.FTE)
- Authority records required for file transfers.

  If you have separate MCAUSER IDs for source and destination agent, create the authority records on agents' queues at both source and destination.

  For example, if the source agent's MCAUSER ID is **`user1`** and the destination agent MCAUSER ID is **`user2`**, set the following authorities for the agent users:

AGENT user	Queue	Authority required
user1	SYSTEM.FTE.DATA.*destination_agent_name*	put
user1	SYSTEM.FTE.COMMAND.*destination_agent_name*	put
user2	SYSTEM.FTE.REPLY.*source_agent_name*	put

AGENT user	Queue	Authority required
user2	SYSTEM.FTE.COMMAND.*source_agent_name*	put

Configuring SSL or TLS between the Connect:Direct bridge agent and the Connect:Direct node

Configure the Connect:Direct bridge agent and the Connect:Direct node to connect to each other through the SSL protocol by creating a keystore and a truststore, and by setting properties in the Connect:Direct bridge agent properties file.

About this task

These steps include instructions for getting your keys signed by a certificate authority. If you do not use a certificate authority, you can generate a self-signed certificate. For more information about generating a self-signed certificate, see "Working with SSL/TLS on AIX, Linux, and Windows" on page 287.

These steps include instructions for creating a new keystore and truststore for the Connect:Direct bridge agent. If the Connect:Direct bridge agent already has a keystore and truststore that it uses to connect securely to IBM MQ queue managers, you can use the existing keystore and truststore when connecting securely to the Connect:Direct node. For more information, see "Configuring SSL or TLS encryption for MFT" on page 574.

Procedure

For the Connect:Direct node, complete the following steps:

1. Generate a key and signed certificate for the Connect:Direct node.

   You can do this by using the IBM Key Management tool that is provided with IBM MQ. For more information, see "Working with SSL/TLS" on page 270.
2. Send a request to a certificate authority to have the key signed. You receive a certificate in return.
3. Create a text file; for example, `/test/ssl/certs/CAcert`, that contains the public key of your certification authority.
4. Install the Secure+ Option on the Connect:Direct node.

   If the node already exists, you can install the Secure+ Option by running the installer again, specifying the location of the existing installation, and choosing to install only the Secure+ Option.
5. Create a new text file; for example, `/test/ssl/cd/keyCertFile/`*`node_name`*`.txt`.
6. Copy the certificate that you received from your certification authority and the private key, located in `/test/ssl/cd/privateKeys/`*`node_name`*`.key`, into the text file.

   The contents of `/test/ssl/cd/keyCertFile/`*`node_name`*`.txt` must be in the following format:

```
-----BEGIN CERTIFICATE-----
MIICnzCCAgigAwIBAgIBGjANBgkqhkiG9w0BAQUFADBeMQswCQYDVQQGEwJHQjES
MBAGA1UECBMJSGFtcHNoaXJlMRAwDgYDVQQHEwdIdXJzbGV5MQwwCgYDVQQKEwNJ
Qk0xDjAMBgNVBAsTBU1RSVBUMQswCQYDVQQDEwJDQTAeFw0xMTAzMDExNjIwNDZa
Fw0yMTAyMjYxNjIwNDZaMFAxCzAJBgNVBAYTAkdCMRIwEAYDVQQIEwlIYW1wc2hp
cmUxDDAKBgNVBAoTA0lCTTEOMAwGA1UECxMFTVFGVEUxDzANBgNVBAMTBmJpbmJh
ZzCBnzANBgkqhkiG9w0BAQEFAAOBjQAwgYkCgYEAvgP1QIklU9ypSKD1XoODo1yk
EyMFXBOUpZRrDVxjoSEC0vtWNcJ199e+Vc4UpNybDyBu+NkDlMNofX4QxeQcLAFj
WnhakqCiQ+JIAD5AurhnrwChe0MV3kjA84GKH/rOSVqtl984mu/lDyS819XcfSSn
cOOMsK1KbneVSCIV2XECAwEAAaN7MHkwCQYDVR0TBAIwADAsBglghkgBhvhCAQ0E
HxYdT3BlblNTTCBHZW5lcmF0ZWQgQ2VydGlmaWNhdGUwHQYDVR0OBBYEFNXMIpSc
csBXUniW4A3UrZnCRsv3MB8GA1UdIwQYMBaAFDXY8rmj4lVz5+FVAoQb++cns+B4
MA0GCSqGSIb3DQEBBQUAA4GBAFc7klXa4pGKYgwchxKpE3ZF6FNwy4vBXS216/ja
8h/vl8+iv01OCL8t0ZOKSU95fyZLzOPKnCH7v+ItFSE3CIiEk9Dlz2U6WO9lICwn
l7PL72TdfaL3kabwHYVf17IVcuL+VZsZ3HjLggP2qHO9ZuJPspeT9+AxFVMLiaAb
8eHw
-----END CERTIFICATE-----
-----BEGIN RSA PRIVATE KEY-----
Proc-Type: 4,ENCRYPTED
DEK-Info: DES-EDE3-CBC,64A02DA15B6B6EF9
```

```
57kqxLOJ/gRUOIQ6hVK2YN13B4E1jAi1gSme0I5ZpEIG8CHXISKB7/0cke2FTqsV
lvI99QyCxsDWoMNt5fj51v7aPmVeS60bOm+UlGre8B/Ze18JVj2O4K2Uh72rDCXE
5e6eFxSdUM207sQDy20euBVELJtM2kOkL1ROdoQQSlU3XQNgJw/t3ZIx5hPXWEQT
rjRQO64BEhb+PzzxPF8uwzZ9IrUK9BJ/UUnqC6OdBR87IeA4pnJD1Jvb2ML7EN9Z
5Y+50hTKI8OGvBvWXO4fHyvIX5aslwhBoArXIS1AtNTrptPvoaP1zyIAeZ6OCVo/
SFo+A2UhmtEJeOJaZG2XZ3H495fAw/EHmjehzIACwukQ9nSIETgu4A1+CV64RJED
aYBCM8UjaAkbZDH5gn7+eBov0ssXAXWDyJBVhUOjXjvAj/e1h+kcSF1hax5D//AI
66nRMZzboSxNqkjcVd8wfDwP+bEjDzUaaarJTS7lIFeLLw7eJ8MNAkMGicDkycL0
EPBU9X5QnHKLKOfYHN/1WgUk8qt3UytFXXfzTXGF3EbsWbBupkT5e5+lYcX8OVZ6
sHFPNlHluCNy/riUcBy9iviVeodX8IomOchSyO5DKl8bwZNjYtUP+CtYHNFU5BaD
I+1uUOAeJ+wjQYKT1WaeIGZ3VxuNITJul8y5qDTXXfX7vxM5OoWXa6U5+AYuGUMg
/itPZmUmNrHjTk7ghT6i1IQOaBowXXKJBlMmq/6BQXN2IhkD9ys2qrvM1hdi5nAf
egmdiG50loLnBRqWbfR+DykpAhK4SaDi2F52Uxovw3Lhiw8dQP7lzQ==
-----END RSA PRIVATE KEY-----
```

7. Start the Secure+ Admin Tool.

   - On AIX and Linux systems, run the command **`spadmin.sh`**.
   - On Windows systems, click **Start** > **Programs** > **Sterling Commerce Connect:Direct** > **CD Secure+ Admin Tool**

   The CD Secure+ Admin Tool starts.
8. In the CD Secure+ Admin Tool, double-click the **.Local** line to edit the main SSL or TLS settings.
   a) Select **Enable SSL Protocol** or **Enable TLS Protocol**, depending on which protocol you are using.
   b) Select **Disable Override**.
   c) Select at least one Cipher Suite.
   d) If you want two-way authentication, change the value of **Enable Client Authentication** to `Yes`.
   e) In the **Trusted Root Certificate** field, enter the path to the public certificate file of your certification authority, `/test/ssl/certs/CAcert`.
   f) In the **Key Certificate File** field, enter the path to the file that you created, `/test/ssl/cd/keyCertFile/`*`node_name`*`.txt`.
9. Double-click the **.Client** line to edit the main SSL or TLS settings.
   a) Select **Enable SSL Protocol** or **Enable TLS Protocol**, depending on which protocol you are using.
   b) Select **Disable Override**.

For the Connect:Direct bridge agent, perform the following steps:

10. Create a truststore. You can do this by creating a dummy key and then deleting the dummy key.

    You can use the following commands:

```
keytool -genkey -alias dummy -keystore /test/ssl/fte/stores/truststore.jks
```

```
keytool -delete -alias dummy -keystore /test/ssl/fte/stores/truststore.jks
```

11. Import the public certificate of the certification authority into the truststore.

    You can use the following command:

```
keytool -import -trustcacerts -alias myCA
        -file /test/ssl/certs/CAcert
        -keystore /test/ssl/fte/stores/truststore.jks
```

12. Edit the Connect:Direct bridge agent properties file.

    Include the following lines anywhere in the file:

```
cdNodeProtocol=protocol
cdNodeTruststore=/test/ssl/fte/stores/truststore.jks
cdNodeTruststorePassword=password
```

    In the example in this step, *protocol* is the protocol you are using, either SSL or TLS, and *password* is the password that you specified when you created the truststore.

13. If you want two-way authentication, create a key and certificate for the Connect:Direct bridge agent.

    a) Create a keystore and key.

       You can use the following command:

       ```
       keytool -genkey -keyalg RSA -alias agent_name
               -keystore /test/ssl/fte/stores/keystore.jks
               -storepass password -validity 365
       ```

    b) Generate a signing request.

       You can use the following command:

       ```
       keytool -certreq -v -alias agent_name
               -keystore /test/ssl/fte/stores/keystore.jks -storepass password
               -file /test/ssl/fte/requests/agent_name.request
       ```

    c) Import the certificate you receive from the preceding step into the keystore. The certificate must be in x.509 format.

       You can use the following command:

       ```
       keytool -import -keystore /test/ssl/fte/stores/keystore.jks
               -storepass password -file certificate_file_path
       ```

    d) Edit the Connect:Direct bridge agent properties file.

       Include the following lines anywhere in the file:

       ```
       cdNodeKeystore=/test/ssl/fte/stores/keystore.jks
       cdNodeKeystorePassword=password
       ```

       In the example in this step, *password* is the password that you specified when you created the keystore.

Related tasks

Configuring the Connect:Direct bridge

ALW Securing AMQP clients

You use a range of security mechanisms to secure connections from AMQP clients and ensure data is suitably protected on the network. You can build security into your MQ Light applications. You can also use existing security features of IBM MQ with AMQP clients, in the same way that the features are used for other applications.

Channel authentication rules (CHLAUTH)

You can use channel authentication rules to restrict the TCP connections to a queue manager. AMQP channels support the use of channel authentication rules that you configure for your queue manager. If channel authentication rules are defined with a profile that matches any AMQP channels on your queue manager, these rules are applied to those channels. By default, channel authentication is enabled on new IBM MQ queue managers so you must complete at least some configuration before you can use an AMQP channel.

For more information about how to configure channel authentication rules to allow AMQP connections to your queue manager, see Creating and using AMQP channels.

Connection authentication (CONNAUTH)

You can use connection authentication to authenticate connections to a queue manager. AMQP channels support the use of connection authentication to control access to the queue manager from AMQP applications.

The AMQP protocol uses the SASL (Simple Authentication and Security Layer) framework to specify how a connection is authenticated. There are various SASL mechanisms and IBM MQ supports two SASL mechanisms: ANONYMOUS and PLAIN.

In the case of ANONYMOUS, no credentials are passed from the client to the queue manager for authentication. If the IBM MQ AUTHINFO object that is specified in the queue manager **CONNAUTH** attribute has a **CHCKCLNT** value of `REQUIRED` or `REQDADM` (if connecting as an administrative user), the connection is refused. If the value of **CHCKCLNT** is `NONE` or `OPTIONAL`, the connection is accepted.

In the case of PLAIN, a user name and password are passed from the client to the queue manager for authentication. If the IBM MQ AUTHINFO object that is specified in the queue manager **CONNAUTH** attribute has a **CHCKCLNT** value of `NONE`, the connection is refused. If the value of **CHCKCLNT** is `OPTIONAL`, `REQUIRED`, or `REQDADM` (if connecting as an administrative user), the user name and password is checked by the queue manager. The queue manager checks the operating system (if the AUTHINFO object is of type `IDPWOS`) or an LDAP repository (if the AUTHINFO object is of type `IDPWLDAP`).

The following table summarizes this authentication behavior:

Table 101. Summary of SASL mechanisms and connection authentication

SASL mechanism	**Credentials passed from client to queue manager?**	**CHKCLNT value**
ANONYMOUS	No	REQUIRED or REQDADM - connection refused NONE or OPTIONAL - connection accepted
PLAIN	Yes, user name and password	REQUIRED, REQDADM, or OPTIONAL - user name and password checked by the queue manager NONE - connection refused

If you are using an MQ Light client, you can specify credentials by including them in the AMQP address you connect to, for example:

```
amqp://mwhitehead:mYp4ssw0rd@localhost:5672/sports/football
```

MCAUSER setting on a channel

AMQP channels have an MCAUSER attribute, which you can use to set the IBM MQ user ID that all connections to that channel are authorized under. All connections from AMQP clients to that channel adopt the MCAUSER ID you have configured. That user ID is used for authorization of messaging on different topics.

You are recommended to use channel authentication (CHLAUTH) to secure connections to queue managers. If you are using channel authentication, you are recommended to configure the value of MCAUSER to a non-privileged user. This ensures that if a connection to a channel is not matched by a CHLAUTH rule, the connection is not authorized to perform any messaging on the queue manager.

SSL/TLS support

AMQP channels support SSL/TLS encryption using keys from the key repository configured for your queue manager. AMQP channel configuration options for SSL/TLS encryption support the same options as other types of MQ channel; you can specify a cipher specification and whether the queue manager requires certificates from AMQP client connections.

By using the FIPS attributes of the queue manager you can control the SSL/TLS cipher suites, which you can use to secure connections from AMQP clients.

For information about how to set up a key repository for the queue manager see "Working with SSL/TLS on AIX, Linux, and Windows" on page 287.

For information about how to configure SSL/TLS support for an AMQP client connection, see Creating and using AMQP channels.

V 9.4.0 V 9.4.0 From IBM MQ 9.4.0, the AMQP channel no longer supports CMS key repositories on the queue manager. You can use the **runmqakm** command to convert a CMS key repository to the PKCS #12 format, which is supported. For example, you can use the following command to convert a key repository named `sslTest.kdb` from CMS format to PKCS #12 format. The new key repository is named `sslTest.p12`, and protected with the password `passw0rd`.

```
runmqakm -keydb -convert -type cms -db sslTest.kdb -stashed -new_format pkcs12 -target
sslTest.p12 -new_pw passw0rd
```

Java Authentication and Authorization Service (JAAS)

You can optionally configure AMQP channels with a JAAS login module, which can check the user name and password provided by an AMQP client. See "Configuring JAAS for AMQP channels" on page 582.

Related tasks

Developing AMQP client applications
Creating and using AMQP channels

ALW Restricting AMQP client takeover

When an AMQP client connection is made that has the same client identifier as an existing AMQP client connection, the existing client connection is disconnected by default. However, you can configure the queue manager to restrict the client takeover behavior so that takeover is possible only when certain criteria are met.

For example, disconnecting the existing client connection might not be appropriate if there are AMQP applications being developed by different teams and they happen to be using the same client ID. To address this issue you can restrict client takeover based on the name of the AMQP channel being used, the IP address of the client, and client User ID (when SASL authentication is enabled).

Use the settings of queue manager attributes **AdoptNewMCA** and **AdoptNewMCACheck** to specify the required level of client takeover restriction, as detailed in the following table:

*Table 102. **AdoptNewMCA** and **AdoptNewMCACheck** settings to restrict client takeover*

AdoptNewMCA	**AdoptNewMCACheck**	**Criteria checked before client takeover is allowed**
NO or undefined	Not applicable	None. Client takeover is allowed for all client connections that are authenticated and pass all CHLAUTH rules.
ALL (or value other than NO)	QM or undefined	None. Client takeover is allowed for all client connections that are authenticated and pass all CHLAUTH rules.
ALL (or value other than NO)	NAME	User ID (when SASL enabled) Channel name

*Table 102. **AdoptNewMCA** and **AdoptNewMCACheck** settings to restrict client takeover (continued)*

AdoptNewMCA	AdoptNewMCACheck	Criteria checked before client takeover is allowed
ALL (or value other than NO)	ADDRESS	User ID (when SASL enabled) IP address
ALL (or value other than NO)	ALL	User ID (when SASL enabled) Channel name IP address

The queue manager attributes **AdoptNewMCA** and **AdoptNewMCACheck** are part of the queue manager configuration, which is defined in the CHANNELS stanza. On IBM MQ for Windows and IBM MQ for Linux x86-64 systems, modify configuration information using the IBM MQ Explorer. On other systems, modify the information by editing the `qm.ini` configuration file. For information about how to modify the queue manager channels information, see Attributes of channels.

Related tasks
Developing AMQP client applications
Creating and using AMQP channels

ALW Configuring JAAS for AMQP channels

Java Authentication and Authorization Service (JAAS) custom modules can be used to authenticate username and password credentials passed to an AMQP channel by an AMQP client when it connects.

About this task

You might want to use a custom JAAS module if you already use JAAS modules for authentication in other Java-based systems, and want to reuse those modules for authenticating AMQP connections to MQ. Alternatively, you might want to write a custom JAAS module if the authentication features built into MQ do not support the authentication mechanism you want to use.

Configuration of JAAS modules for AMQP channels is done at a queue manager level. This means that, if you configure a JAAS module for authenticating AMQP connections to the queue manager, the module will apply to all AMQP channels. The name of the channel that has invoked the JAAS module is passed to the module, allowing you to code different JAAS log in behavior for different channels.

Other information is also passed the JAAS module:

- The client ID of the AMQP client that is attempting to authenticate.
- The network address of the AMQP client.
- The name of the channel that invoked the JAAS module.

Procedure

You configure a JAAS configuration module for AMQP channels by completing the following steps:

1. Define a `jaas.config` file containing one or more JAAS module configuration stanzas. The stanza must specify the fully qualified name of the Java class that implements the JAAS `javax.security.auth.spi.LoginModule` interface.
    - A default `jaas.config` file is shipped with the product and is located in *`QM_data_directory`*`/amqp/jaas.config`.
    - A preconfigured stanza named `MQXRConfig` is already defined in the default `jaas.config` file.
2. Specify the name of the stanza to use for AMQP channels.
    - Linux AIX Add a property to the `amqp_unix.properties` file.

- Windows Add a property to the `amqp_win.properties` file.

  The property has the following form:

  ```
  com.ibm.mq.MQXR.JAASConfig=JAAS_stanza_name
  ```

  for example:

  ```
  com.ibm.mq.MQXR.JAASConfig=MQXRConfig
  ```

3. Configure the queue manager environment to include the class of the custom module. The AMQP service must have access to the Java class configured in the JAAS configuration stanza.

   You do this by adding the path to the JAAS class to the MQ `service.env` file. Edit the `service.env` file in the MQ configuration directory (*MQ_config_directory*) or the queue manager configuration directory (*QM_config_directory*) to set the CLASSPATH variable to the location of the JAAS module class.

What to do next

A sample JAAS login module is shipped with the product in the *`mq_installation_directory`*`/amqp/samples` directory. The sample JAAS login module authenticates all client connections, regardless of the username or password the client connects with.

You can modify the source code of the sample and recompile it to try authenticating only specific users with a particular password. To configure the AMQP channel on a UNIX system to use the sample JAAS login module shipped with the product:

1. Edit the file `/var/mqm/qmgrs/QMNAME/amqp/amqp_unix.properties` and set the property `com.ibm.mq.MQXR.JAASConfig=MQXRConfig`.
2. Edit the file /var/mqm/service.env and set the property `CLASSPATH=`*`mq_installation_location`*`/amqp/samples`

The `jaas.config` file already contains a stanza named `MQXRConfig` that specifies the sample class `samples.JAASLoginModule` as the login module class. No changes are required to `jaas.config` before you try the sample module.

Related tasks

Developing AMQP client applications

Creating and using AMQP channels

Advanced Message Security

Advanced Message Security (AMS) is a component of IBM MQ that provides a high level of protection for sensitive data flowing through the IBM MQ network, while not impacting the end applications.

About this task

Advanced Message Security expands IBM MQ security services to provide data signing and encryption at the message level. The expanded services guarantee that message data has not been modified between when it is originally placed on a queue and when it is retrieved. In addition, AMS verifies that a sender of message data is authorized to place signed messages on a target queue.

AMS provides the following functions:

- Secures sensitive or high-value transactions processed by IBM MQ.
- Detects and removes rogue or unauthorized messages before they are processed by a receiving application.
- Verifies that messages were not modified while in transit from queue to queue.
- Protects the data not only as it flows across the network but also when it is put on a queue.

- Secures existing proprietary and customer-written applications for IBM MQ.
- z/OS From IBM MQ 9.1.3, IBM MQ for z/OS provides the ability to optionally remove and add AMS protection from, or to, messages that flow across the network, respectively. This is known as *Server to Server Message Channel Agent (MCA) Interception.*.
- ALW From IBM MQ 9.1.4 and IBM MQ 9.1.0 Fix Pack 4, a check is added to the IBM MQ library code that runs within the customer's application program. The check runs early in its initialization to read the value of the environment variable *AMQ_AMS_FIPS_OFF* and, if it is set to any value, then the IBM Global Security Kit (GSKit) code is run in non-FIPS mode in that application.

Procedure

Overview of Advanced Message Security

IBM MQ applications can use Advanced Message Security to send sensitive data, such as high-value financial transactions and personal information, with different levels of protection by using a public key cryptography model.

Related concepts

"Message Channel Agent (MCA) interception and AMS" on page 633

MCA interception enables a queue manager running under IBM MQ to selectively enable policies to be applied for server connection channels.

Related reference

GSKit return codes used in AMS messages

Qualities of protection available with AMS

There are three qualities of protection for Advanced Message Security, `Integrity`, `Privacy`, and `Confidentiality`.

`Integrity` protection is provided by digital signing, which provides assurance on who created the message, and that the message has not been altered or tampered with.

`Privacy` protection is provided by a combination of digital signing and encryption. Encryption ensures that message data is only viewable to the intended recipient, or recipients. Even if unauthorized recipients obtain a copy of the encrypted message data, they are unable to view the actual message data itself.

`Confidentiality` protection is provided by encryption only with optional key reuse.

Effect on performance

AMS uses a combination of symmetric and asymmetric cryptographic routines to provide digital signing and encryption. As symmetric key operations are very fast in comparison to asymmetric key operations, which are CPU intensive, this in turn can have a significant impact on the costs of protecting large numbers of messages with AMS.

Asymmetric cryptographic routines

For example, when putting a signed message, the message hash is signed using an asymmetric key operation.

When getting a signed message, a further asymmetric key operation is used to verify the signed hash.

Therefore, a minimum of two asymmetric key operations are required per message to sign and verify the message data.

Asymmetric and symmetric cryptographic routines

When putting an encrypted message, a symmetric key is generated and then encrypted using an asymmetric key operation for each intended recipient of the message.

The message data is then encrypted with the symmetric key. When getting the encrypted message the intended recipient needs to use an asymmetric key operation to discover the symmetric key in use for the message.

All three qualities of protection, therefore, contain varying elements of the CPU intensive asymmetric key operations, which will significantly impact the maximum achievable messaging rate for applications putting and getting messages.

`Confidentiality` policies do, however, allow for symmetric key reuse over a sequence of messages. Significant CPU cost savings can be made with `Confidentiality` policies through symmetric key reuse. This mode of operation continues to use the PKCS#7 format to share a symmetric encryption key. However, there is no digital signature, which eliminates some of the per message asymmetric key operations. The symmetric key still needs to be encrypted with asymmetric key operations for each recipient, but the symmetric key can be optionally reused over multiple messages that are destined for the same recipients. If key reuse is permitted by policy, then only the first message requires asymmetric key operations. Subsequent messages only need to use symmetric key operations.

Key reuse

With `Confidentiality` policies, you can use the symmetric key reuse approach to significantly reduce the costs involved in encrypting a number of messages that are put to the same queue and intended for the same recipient or recipients.

For example, when putting 10 encrypted messages to the same set of recipients, a symmetric key is generated, and then encrypted for the first message, using an asymmetric key operation for each intended recipient of the message.

Based upon policy controlled limits, the encrypted symmetric key can then be reused by subsequent messages that are intended for the same recipients. To allow the symmetric key to be reused by subsequent messages, the application must keep the queue open after putting a message to the queue. The symmetric key cannot be reused by MQPUT1 operations. An application that is getting encrypted messages can apply the same optimization, in that the application can detect when a symmetric key has not changed and avoid the expense of retrieving the symmetric key.

In this example 90% of the asymmetric key operations can be avoided by both the putting and getting applications by reusing the same key.

For more information on how to use key reuse, see:

- MQSC command SET POLICY
- Control command setmqspl
- IBM i IBM i command SETMQMSPL

Key concepts in AMS

Learn about the key concepts in Advanced Message Security to understand how the tool works and how to manage it effectively.

Public key infrastructure and Advanced Message Security

Public key infrastructure (PKI) is a system of facilities, policies, and services that support the use of public key cryptography to obtain secure communication.

There is no single standard that defines the components of a public key infrastructure, but a PKI typically involves usage of public key certificates and comprises certificate authorities (CA) and other registration authorities (RA) that provide the following services:

- Issuing digital certificates
- Validating digital certificates
- Revoking digital certificates
- Distributing certificates

Identity of users and applications are represented by the **distinguished name (DN)** field in a certificate associated with signed or encrypted messages. Advanced Message Security uses this identity to represent a user or an application. To authenticate this identity, the user or application must have access to the keystore where the certificate and associated private key are stored. Each certificate is represented by a label in the keystore.

Related concepts

"Using keystores and certificates with AMS" on page 627
To provide transparent cryptographic protection to IBM MQ applications, Advanced Message Security uses the keystore file, where public key certificates and a private key are stored. On z/OS, a SAF key ring is used instead of a keystore file.

Digital certificates in AMS

Advanced Message Security associates users and applications with X.509 standard digital certificates. X.509 certificates are typically signed by a trusted certificate authority (CA) and involve private and public keys which are used for encryption and decryption.

Digital certificates provide protection against impersonation by binding a public key to its owner, whether that owner is an individual, a queue manager, or some other entity. Digital certificates are also known as public key certificates, because they give you assurance about the ownership of a public key when you use an asymmetric key scheme. This scheme requires that a public key and a private key be generated for an application. Data encrypted with the public key can only be decrypted using the corresponding private key while data encrypted with the private key can only be decrypted using the corresponding public key. The private key is stored in a key database file that is password-protected. Only its owner has the access to the private key used to decrypt messages that are encrypted using the corresponding public key.

If public keys are sent directly by their owner to another entity, there is a risk that the message could be intercepted and the public key substituted by another. This is known as a "man-in-the-middle" attack. The solution is to exchange public keys through a trusted third party, giving the user a strong assurance that the public key belongs to the entity with which you are communicating. Instead of sending your public key directly, you ask a trusted third party to incorporate it into a digital certificate. The trusted third-party who issues digital certificates is called a certificate authority (CA).

For more information about digital certificates, see What is in a digital certificate.

A digital certificate contains the public key for an entity and states that the public key belongs to that entity:

- when a certificate is for an individual entity, it is called a *personal certificate* or *user certificate*.
- when a certificate is for a certificate authority, the certificate is called a *CA certificate* or *signer certificate*.

Note: Advanced Message Security supports self-signed certificates in both Java and native applications

Related concepts

"Cryptography" on page 11
Cryptography is the process of converting between readable text, called *plaintext*, and an unreadable form, called *ciphertext*.

Multi *Object authority manager and AMS*

On Multiplatforms, the Object Authority Manager (OAM) is the authorization service component supplied with the IBM MQ products.

The access to Advanced Message Security entities is controlled through IBM MQ user groups and the OAM. Administrators can use the command-line interface to grant or revoke authorizations as required. Different groups of users can have different kinds of access authority to the same objects. For example, one group could perform both PUT and GET operations for a specific queue while another group might be allowed only to browse the queue. Similarly, some groups might have GET and PUT authority to a queue, but are not allowed to alter or delete the queue.

Through the OAM, you can control:

- Access to Advanced Message Security objects through the Message Queue Interface (MQI). When an application program attempts to access objects, the OAM checks if the user profile making the request has the authorization for the operation requested. This means that queues, and the messages on queues, can be protected from unauthorized access.

- Permission to use PCF and MQSC commands.

Related concepts

Object authority manager

The Message Queue Interface overview

Technology supported by Advanced Message Security

Advanced Message Security depends on several technology components to provide a security infrastructure.

Advanced Message Security supports the following IBM MQ application programming interfaces (APIs):

- Message queue interface (MQI)
- IBM MQ Java Message Service (JMS) 1.0.2 and 1.1.
- IBM MQ Base Classes for Java
- IBM MQ classes for .Net in an unmanaged mode

Note: Advanced Message Security supports X.509 compliant certificate authorities.

Known limitations of AMS

There are a number of IBM MQ options that are either not supported, or have limitations for Advanced Message Security.

- The following IBM MQ options are not supported or have limitations:

  Publish/subscribe

  One of the major benefits of a publish/subscribe messaging model over point-to-point is that the sending and receiving applications do not need to know anything about each other for data to be sent and received. This benefit is negated by the use of Advanced Message Security policies that must define intended recipients or authorized signers. It is possible for an application to publish to a topic via an alias queue definition that is protected by a policy, it is also possible for a subscribing application to get messages from a policy protected queue. It is not possible to assign a policy directly to a topic string, policies can only be assigned to queue definitions.

  Channel data conversion

  The protected payload of an Advanced Message Security protected message is transmitted using binary format, this ensures that data conversion on a channel between applications does not invalidate the message digest. Applications retrieving messages from a policy protected queue should request data conversion, the conversion of the protected payload will be attempted after messages have been successfully verified and unprotected.

  Distribution lists

  Advanced Message Security policies can be used when protecting applications putting messages to distribution lists, provided each destination queue in the list has an identical policy defined. If inconsistent policies are identified when an application opens a distribution list, the open operation will fail and a security error returned to the application.

  Application message segmentation

  The size of policy protected messages will increase and it is not possible for applications to accurately specify the segment boundaries of a message.

  Applications using IBM MQ classes for .NET in a managed mode (client connections)

  Applications using IBM MQ classes for .NET in a managed mode (client connections) are not supported.

  Note: MCA interception can be used to allow unsupported clients to use AMS.

  Message Service client for .NET (XMS) applications in a managed mode

  Message Service client for .NET (XMS) applications in a managed mode are not supported.

Note: MCA interception can be used to allow unsupported clients to use AMS.

IBM MQ queues processed by the IMS bridge
IBM MQ queues processed by the IMS bridge are not supported.

Note: AMS is supported on CICS bridge queues. You should use the same user ID to MQPUT (encrypt) and MQGET (decrypt) on CICS bridge queues.

Put to waiting getter
Put to waiting getter is not supported for getter applications against queues that have AMS policies defined for them.

z/OS **Server to server MCA interception**
From IBM MQ for z/OS 9.1.3, server to server MCA interception is only supported for sender, server, receiver and requestor channel types.

- Users should avoid putting more than one certificate with the same Distinguished Name in a single keystore file, because the choice of which certificate to use when protecting a message is undefined.
- AMS is not supported in JMS if the **`WMQ_PROVIDER_VERSION`** property is set to 6.
- The AMS interceptor is not supported for AMQP or MQTT channels.

z/OS Advanced Message Security interception on message channels

On z/OS, Advanced Message Security (AMS) interception provides an additional option of security policy protection (SPLPROT) to sender, server, receiver, and requester channels, allowing you to support AMS and to communicate with business partners who do not support AMS.

Taking the example of a clearing house communicating with a bank, Figure 1 shows that, without AMS interception, both sides of the system need to support AMS.

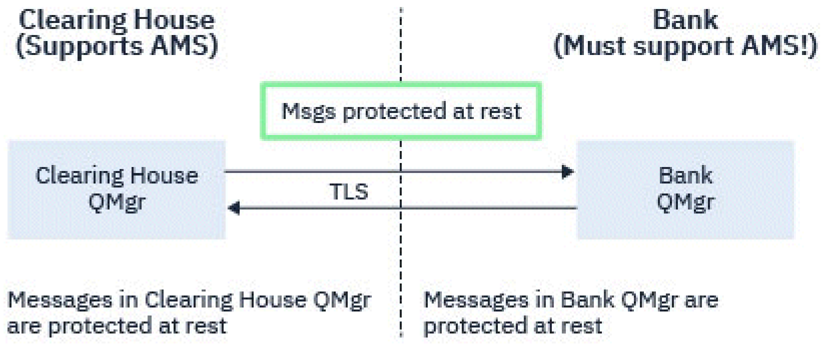

Clearing House (Supports AMS)
Bank (Must support AMS!)
Msgs protected at rest
Clearing House QMgr
TLS
Bank QMgr
Messages in Clearing House QMgr are protected at rest
Messages in Bank QMgr are protected at rest

Figure 32. Usage of AMS without AMS interception

A key benefit of the AMS interception option is, that if your enterprise has AMS configured, and not all of your business partners support AMS, you can remove protection from outbound messages and protect inbound messages on channels to and from those business partners that do not support AMS.

Using the example of a clearing house and banks, this scenario is shown in Figure 2, where there is a message flow between the clearing house, banks, and business partners where some institutions have AMS, and others do not.

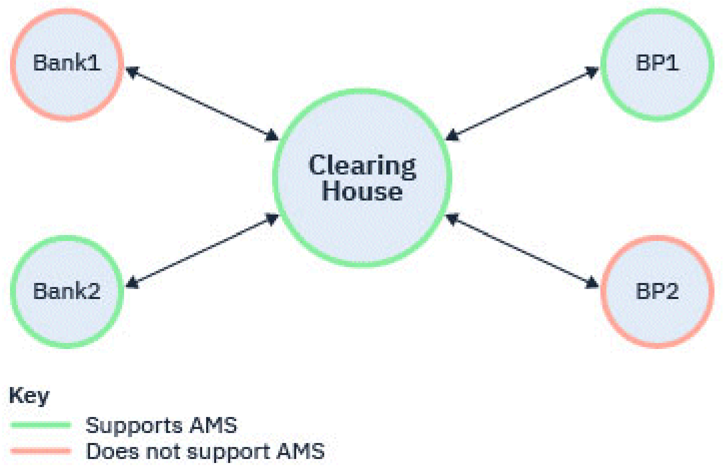


Figure 33. Some partners support AMS and some do not

Typically the channels are TLS enabled.

However, there might be a case where some banks and business partners do not support AMS, and there is a requirement to be able to exchange messages between all banks and business partners. This scenario is shown in Figure 3

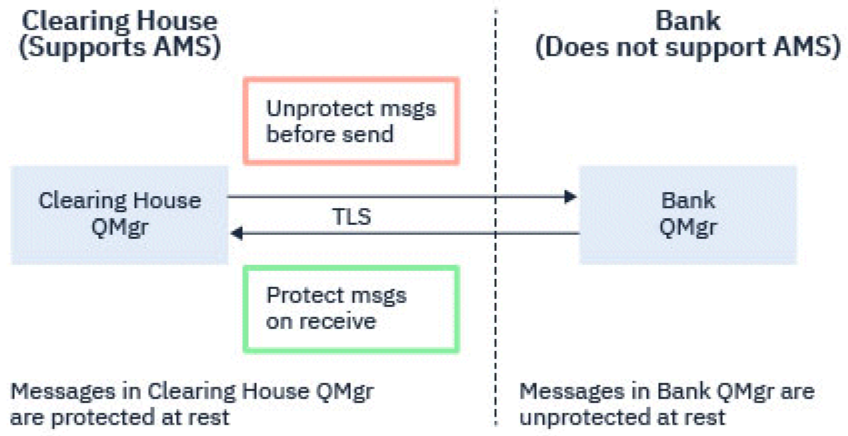

Clearing House (Supports AMS)
Bank (Does not support AMS)
Unprotect msgs before send
Clearing House QMgr
TLS
Bank QMgr
Protect msgs on receive
Messages in Clearing House QMgr are protected at rest
Messages in Bank QMgr are unprotected at rest

Figure 34. Message flow between business partners

Related tasks

Server-to-server message channel interception example configurations

z/OS *AMS interception on server-to-server message channels*

Server-to-server message channel interception provides a means to control if messages should have any applicable Advanced Message Security (AMS) policies applied to them, when sender type message channel agents get messages from transmission queues, and receiver type message channel agents put messages to target queues.

This allows AMS protection to be enabled on a queue manager when communicating, using server-to-server message channels of type sender, server, receiver, and requester, with a queue manager that does not have AMS enabled.

That is, AMS protected messages in AMS enabled queue managers can be unprotected prior to being sent to non-AMS enabled queue managers, and unprotected messages received from non-AMS enabled queue managers can be protected, by applicable AMS policies, on AMS enabled queue managers.

Configuring server-to-server message channel interception

Server-to-server message channel interception is configured with the SPLPROT attribute on channels with a channel type of sender, server, receiver, or requester. The available options to configure the behavior are dependent on the channel type specified:

PASSTHRU
Pass through, unchanged, any messages sent or received by the message channel agent for this channel.

This value is valid for channels with a channel type (**CHLTYPE**) of SDR, SVR, RCVR, or RQSTR, and is the default value.

REMOVE
Remove any AMS protection from messages retrieved from the transmission queue by the message channel agent, and send the messages to the partner.

When the message channel agent gets a message from the transmission queue, if an AMS policy is defined for the transmission queue, it is applied to remove any AMS protection from the message prior to sending the message across the channel. If an AMS policy is not defined for the transmission queue, the message is sent as is.

This value is valid only for channels with a channel type of SDR or SVR.

ASPOLICY
Based on the policy defined for the target queue, apply AMS protection to inbound messages prior to putting them on to the target queue.

When the message channel agent receives an inbound message, if an AMS policy is defined for the target queue, AMS protection is applied to the message prior to the message being put to the target queue. If an AMS policy is not defined for the target queue, the message is put to the target queue as is.

This value is valid only for channels with a channel type of RCVR or RQSTR.

User ID for message channel interception

The requirement for user IDs used with server-to-server message channel interception are the same as those for existing AMS enabled applications. For a running channel, the sending message channel agent gets messages from a transmission queue and the receiving message channel agent puts messages to target queues. The message channel agent user ID (MCAUSER) field, set on server to server channels, defines the user ID under which message channel agents perform put and get requests.

With server-to-server message channel interception, AMS functions are performed during get and put requests, as with other AMS enabled applications. Therefore, message channel agent user Ids have the same requirements as those for AMS application user IDs.

The MCAUSER used to perform the put and get is configurable, and dependent on whether it is an outbound or inbound channel. See MCAUSER for details of how the chosen user ID performs actions on the message channel agent. As such, the user ID that the channel initiator is running under is the user ID that is to be used for AMS functions performed during server-to-server message channel interception. Therefore, these user IDs have the same requirements as those for AMS application user IDs.

Authentication is performed using the existing rules for the channel detailed for channels with PUTAUT configuration. See user IDs used by the channel initiator for more information.

Note: Server-to-server message channel interception does not take into account the value of the PUTAUT channel attribute.

Message size and MAXMSGL

Due to AMS protection, the message size of protected messages will be larger than the original message size.

Protected messages are larger than unprotected messages. Therefore, the value of the **MAXMSGL** attribute, on both queues and channels, might need to be altered to take into account the size of protected messages.

Related reference

Server-to-server message channel interception example configurations

Error handling for AMS

IBM MQ Advanced Message Security defines an error handling queue to manage messages that contain errors or messages that cannot be unprotected.

Defective messages are dealt with as exceptional cases. If a received message does not meet the security requirements for the queue it is on, for example, if the message is signed when it should be encrypted, or decryption or signature verification fails, the message is sent to the error handling queue. A message might be sent to the error handling queue for the following reasons:

- Quality of protection mismatch - a quality of protection (QOP) mismatch exists between the received message and the QOP definition in the security policy.
- Decryption error - the message cannot be decrypted.
- PDMQ header error - the Advanced Message Security (AMS) message header cannot be accessed.
- Size mismatch - length of a message after decryption is different than expected.
- Encryption algorithm strength mismatch - the message encryption algorithm is weaker than required.
- Unknown error - unexpected error occurred.

AMS uses the SYSTEM.PROTECTION.ERROR.QUEUE as its error handling queue. All messages put by IBM MQ AMS to the SYSTEM.PROTECTION.ERROR.QUEUE are preceded by an MQDLH header.

Your IBM MQ administrator can also define the SYSTEM.PROTECTION.ERROR.QUEUE as an alias queue pointing to another queue.

z/OS On IBM MQ for z/OS, if server to server Message Channel Agent (MCA) interception is in use:

- If, for one of the previously stated reasons, IBM MQ AMS moves messages from the transmission queue to the error handling queue, the sender MCA simply proceeds to process the next available message on the transmission queue.
- In general, existing channel rules apply for:
  - Putting messages to the Dead Letter Queue, and
  - Actions taken if puts to the Dead Letter Queue should fail.

See "Undelivered messages for AMS on z/OS" on page 591 for further information on specific scenarios.

z/OS *Undelivered messages for AMS on z/OS*

Specific scenarios related to server to server Message Channel Agent interception on IBM MQ for z/OS.

On IBM MQ for z/OS, if server to server Message Channel Agent (MCA) interception is in use:

- If, after having got and unprotected a message, the sender MCA fails to deliver a message for some reason, for example, because the message is too big for the channel, if the USEDLQ sender channel attribute is set to YES, the sender MCA moves the message to the local Dead Letter Queue (DLQ).

  If the SYSTEM.DEAD.LETTER.QUEUE is being used as the local DLQ, the message is placed unprotected.

  Note: IBM MQ AMS does not support the protection of messages put to system queues.

  If a named DLQ is being used as the local DLQ, the message will be placed protected if you have defined an IBM MQ AMS policy with the same name as the named DLQ, and unprotected if you have not defined a suitable policy.
- If a message cannot be put to the local DLQ for some reason, then if the NPMSPEED of the channel is set to NORMAL, or the message is a persistent message, the current batch of messages is backed

out and the channel put into RETRY state. Otherwise, the message is discarded and the sender MCA continues to process the next message on the transmission queue.

- Given that security policies have no effect on the SYSTEM.DEAD.LETTER.QUEUE, or the other SYSTEM queues listed in "System queue protection in AMS" on page 663, if the SYSTEM.DEAD.LETTER.QUEUE is in use, messages put to this queue by MCAs are placed as is. That is, if messages were previously protected, they are placed protected; otherwise, they are placed unprotected.

  If the queue manager DEADQ attribute has been set to the name of an alternate (non-system) dead letter queue, and an AMS policy with the same name does not exist, messages put to this queue by MCAs are placed as is. That is, if messages were previously protected, they are placed protected; otherwise, they are placed unprotected.

  If the queue manager DEADQ attribute has been set to the name of an alternate (non-system) dead letter queue and an AMS policy with the same name as the DLQ does exist, the policy is used to protect messages put to this queue by MCAs. If the message has previously already been protected, it is not protected again; this is to avoid double protection. If an AMS policy with the same name does not exist, messages are placed as-is.
- If there is a policy for the DLQ with the tolerate option in the setmqspl command set to off, that is '`-t` O', the put to the DLQ fails if the message is not AMS protected, and hence does not have a PDMQ header. This happens if the message arrives at the receiver without a PDMQ header. That is the original putter of the message did not have a policy for the destination, and the receiver does not have SPLPROT(ASPOLICY) set.
- An MCA might fail to put a message to the DLQ, if the AMS policy defined for the DLQ does not permit the user ID that the channel initiator is running under to protect the message.
- Receiver channels generally place undelivered messages to the local DLQ, while sender channels generally place messages that cannot be processed for some reason, for example, message too large for queue, or bad MQXQH header, and so on to the local DLQ.
- DLQ handlers generally only look at the DLQ header (DLH) and not the message payload itself. So, the fact that the message payload might be protected, does not prevent handlers from determining why the message was placed on the DLQ.
- If a DLQ is not defined, the channel:
  - Ends abnormally (and goes into retrying state) if a persistent message cannot be delivered.
  - Discards a non-persistent undelivered message, and continues to run.

Related concepts
"Error handling for AMS" on page 591
IBM MQ Advanced Message Security defines an error handling queue to manage messages that contain errors or messages that cannot be unprotected.

User scenarios for AMS

Familiarize yourself with possible scenarios to understand what business goals you can achieve with Advanced Message Security.

Windows *Quick Start Guide for AMS on Windows platforms*

Use this guide to quickly configure Advanced Message Security (AMS) to provide message security on Windows platforms. By the time you complete it, you will have created a key database to verify user identities, and defined signing/encryption policies for your queue manager.

Before you begin

You should have at least the following features installed on your system:

- Server
- Development Toolkit (for the Sample programs)
- Advanced Message Security (AMS)

Refer to IBM MQ features for Windows systems for details.

For information about using the **setmqenv** command to initialize the current environment so that the appropriate IBM MQ commands can be located and executed by the operating system, see setmqenv (set IBM MQ environment).

1. Creating a queue manager and a queue

About this task

All the following examples use a queue named `TEST.Q` for passing messages between applications. Advanced Message Security uses interceptors to sign and encrypt messages at the point they enter the IBM MQ infrastructure through the standard IBM MQ interface. The basic setup is done in IBM MQ and is configured in the following steps.

You can use IBM MQ Explorer to create the queue manager QM_VERIFY_AMS and its local queue called `TEST.Q` by using all the default wizard settings, or you can use the commands found in `C:\Program Files\IBM\MQ\bin`. Remember that you must be a member of the `mqm` user group to run the following administrative commands.

Procedure

1. Create a queue manager

   ```
   crtmqm QM_VERIFY_AMS
   ```

2. Start the queue manager

   ```
   strmqm QM_VERIFY_AMS
   ```

3. Create a queue called `TEST.Q` by entering the following command into **runmqsc** for queue manager `QM_VERIFY_AMS`

   ```
   DEFINE QLOCAL(TEST.Q)
   ```

Results

If the procedure is completed, command entered into **runmqsc** will display details about `TEST.Q`:

```
DISPLAY Q(TEST.Q)
```

2. Creating and authorizing users

About this task

There are two users that appear in this example: `alice`, the sender, and bob, the receiver. To use the application queue, these users need to be granted authority to use it. Also to successfully use the protection policies that we will define these users must be granted access to some system queues. For more information about the **setmqaut** command refer to **setmqaut**.

Procedure

1. Create the two users and ensure that `HOMEPATH` and `HOMEDRIVE` are set for both these users.
2. Authorize the users to connect to the queue manager and to work with the queue

   ```
   setmqaut -m QM_VERIFY_AMS -t qmgr -p alice -p bob +connect +inq
   ```

```
setmqaut -m QM_VERIFY_AMS -n TEST.Q -t queue -p alice +put
```

```
setmqaut -m QM_VERIFY_AMS -n TEST.Q -t queue -p bob +get
```

3. You should also allow the two users to browse the system policy queue and put messages on the error queue.

```
setmqaut -m QM_VERIFY_AMS -t queue -n SYSTEM.PROTECTION.POLICY.QUEUE -p alice -p bob +browse
```

```
setmqaut -m QM_VERIFY_AMS -t queue -n SYSTEM.PROTECTION.ERROR.QUEUE -p alice -p bob +put
```

Attention: IBM MQ optimizes performance by caching policies so that you do not have to browse records for policy details on the SYSTEM.PROTECTION.POLICY.QUEUE in all cases.

IBM MQ does not cache all the policies available. If there are high number of policies, IBM MQ caches a limited number of policies. So, if the queue manager has a low number of policies defined, there is no need to provide the browse option to the SYSTEM.PROTECTION.POLICY.QUEUE.

However, you should give browse authority to this queue, in case there is a high number of policies defined, or if you are using old clients. The SYSTEM.PROTECTION.ERROR.QUEUE is used to put error messages generated by the AMS code. The put authority against this queue is checked only when you attempt to put an error message to the queue. Your put authority against the queue is not checked when you attempt to put or get message from an AMS protected queue.

Results

Users are now created and the required authorities granted to them.

What to do next

To verify if the steps were carried out correctly, use the `amqsput` and `amqsget` samples as described in section "7. Testing the setup" on page 597.

3. Creating key database and certificates

About this task

Interceptor requires the public key of the sending users to encrypt the message. Thus, the key database of user identities mapped to public and private keys must be created. In the real system, where users and applications are dispersed over several computers, each user would have its own private keystore. Similarly, in this guide, we create key databases for `alice` and bob and share the user certificates between them.

Note: In this guide, we use sample applications written in C connecting using local bindings. If you plan to use Java applications using client bindings, you must create a JKS keystore and certificates using either the Java **`keytool`** command V 9.4.0 V 9.4.0 or the IBM MQ **`runmqktool`** command. For more information, see "Quick Start Guide for AMS with Java clients" on page 614. For all other languages, and for Java applications using local bindings, the steps in this guide are correct.

Procedure

1. Create a new key database for the user `alice`.
   For example, issue the following command to create the new key database:

```
runmqakm -keydb -create -db "C:\Documents and Settings\alice\AMS\alicekey.kdb" -type cms -pw
passw0rd -stash
```

Note:

- Use a strong password to secure the database.
- Include the **-stash** parameter to stash the encrypted key database password in a file.

2. Create a new self-signed certificate to identify the user `alice` for use in encryption.
   For example, issue the following command to create a new self-signed certificate:

```
runmqakm -cert -create -db "C:\Documents and Settings\alice\AMS\alicekey.kdb" -stashed
-label Alice_Cert -dn "CN=alice, O=IBM, C=GB"
```

   Note:

   - For the purpose of this guide, we are using self-signed certificate which can be created without using a Certificate Authority. For production systems, it is advisable to use certificates that are signed by a Certificate Authority.
   - 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) for the certificate. The Distinguished Name must be unique for each user.

3. Repeat steps “1” on page 594 and “2” on page 595 for the user bob.

Results

The two users `alice` and bob each now have a self-signed certificate.

4. Creating keystore.conf

About this task

You must point Advanced Message Security interceptors to the directory where the key databases and certificates are located. This is done via the `keystore.conf` file, which holds that information in plain text form. Each user must have a separate `keystore.conf` file in the `.mqs` folder. This step must be done for both `alice` and bob.

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

```
cms.keystore = dir/keystore_file
cms.certificate = certificate_label
```

Example

For this scenario, the contents of the `keystore.conf` will be as follows:

```
cms.keystore = C:/Documents and Settings/alice/AMS/alicekey
cms.certificate = Alice_Cert
```

Note:

- The path to the keystore file must be provided with no file extension.
- The certificate label can include spaces, thus "Alice_Cert" and "Alice_Cert " (with a space on the end) for example, are recognized as labels of two different certificates. However, to avoid confusion, it is better not to use spaces in label's name.
- There are the following keystore formats: CMS (Cryptographic Message Syntax), JKS (Java Keystore) and JCEKS (Java Cryptographic Extension Keystore). For more information, refer to “Structure of the keystore configuration file (keystore.conf) for AMS” on page 628.
- *%HOMEDRIVE%*`\`*%HOMEPATH%*`\.mqs\keystore.conf` (eg. C:\Documents and Settings\alice\.mqs\keystore.conf) is the default location where Advanced Message Security searches for the `keystore.conf` file. For information about how to use a non-default location for the `keystore.conf`, see “Using keystores and certificates with AMS” on page 627.

- To create `.mqs` directory, you must use the command prompt.

5. Sharing Certificates

About this task

Share the certificates between the two key databases so that each user can successfully identify the other. This is done by extracting each user's public certificate to a file, which is then added to the other user's key database.

Note: Take care to use the *extract* option, and not the *export* option. *Extract* gets the user's public key, whereas *export* gets both the public and private key. Using *export* by mistake would completely compromise your application, by passing on its private key.

Procedure

1. Extract the certificate identifying `alice` to an external file:

```
runmqakm -cert -extract -db "C:\Documents and Settings\alice\AMS\alicekey.kdb" -pw passw0rd
-label Alice_Cert -target alice_public.arm
```

2. Add the certificate to `bob's` keystore:

```
runmqakm -cert -add -db "C:\Documents and Settings\bob\AMS\bobkey.kdb" -pw passw0rd -label
Alice_Cert -file alice_public.arm
```

3. Repeat steps for bob:

```
runmqakm -cert -extract -db "C:\Documents and Settings\bob\AMS\bobkey.kdb" -pw passw0rd
-label Bob_Cert -target bob_public.arm
```

```
runmqakm -cert -add -db "C:\Documents and Settings\alice\AMS\alicekey.kdb" -pw passw0rd
-label Bob_Cert -file bob_public.arm
```

Results

The two users `alice` and bob are now able to successfully identify each other having created and shared self-signed certificates.

What to do next

Verify that a certificate is in the keystore either by browsing it using the GUI or running the following commands which print out its details:

```
runmqakm -cert -details -db "C:\Documents and Settings\bob\AMS\bobkey.kdb" -pw passw0rd -label
Alice_Cert
```

```
runmqakm -cert -details -db "C:\Documents and Settings\alice\AMS\alicekey.kdb" -pw passw0rd
-label Bob_Cert
```

6. Defining queue policy

About this task

With the queue manager created and interceptors prepared to intercept messages and access encryption keys, we can start defining protection policies on `QM_VERIFY_AMS` using the `setmqspl` command. Refer to setmqspl for more information on this command. Each policy name must be the same as the queue name it is to be applied to.

Example

This is an example of a policy defined for the `TEST.Q` queue. In the example, messages are signed with the Deprecated SHA1 algorithm and encrypted with the AES256 algorithm. `alice` is the only valid sender and bob is the only receiver of the messages on this queue:

```
setmqspl -m QM_VERIFY_AMS -p TEST.Q -s SHA1 -a "CN=alice,O=IBM,C=GB" -e AES256 -r
"CN=bob,O=IBM,C=GB"
```

Note: The DNs match exactly those specified in the respective user's certificate from the key database.

What to do next

To verify the policy you have defined, issue the following command:

```
dspmqspl -m QM_VERIFY_AMS
```

To print the policy details as a set of `setmqspl` commands, use the `-export` flag. This allows storing already defined policies:

```
dspmqspl -m QM_VERIFY_AMS -export >restore_my_policies.bat
```

7. Testing the setup

About this task

By running different programs under different users you can verify if the application has been properly configured.

Procedure

1. Switch user to run as user `alice`

   Right-click `cmd.exe` and select **Run as...**. When prompted, log in as the user `alice`.
2. As the user `alice` put a message using a sample application:

   ```
   amqsput TEST.Q QM_VERIFY_AMS
   ```
3. Type the text of the message, then press Enter.
4. Switch user to run as user bob

   Open another window by right-clicking `cmd.exe` and selecting **Run as...**. When prompted, log in as the user bob.
5. As the user bob get a message using a sample application:

   ```
   amqsget TEST.Q QM_VERIFY_AMS
   ```

Results

If the application has been configured properly for both users, the user `alice` 's message is displayed when bob runs the getting application.

8. Testing encryption

About this task

To verify that the encryption is occurring as expected, create an alias queue which references the original queue `TEST.Q`. This alias queue will have no security policy and so no user will have the information to decrypt the message and therefore the encrypted data will be shown.

Procedure

1. Using the **runmqsc** command against queue manager QM_VERIFY_AMS, create an alias queue.

```
DEFINE QALIAS(TEST.ALIAS) TARGET(TEST.Q)
```

2. Grant bob access to browse from the alias queue

```
setmqaut -m QM_VERIFY_AMS -n TEST.ALIAS -t queue -p bob +browse
```

3. As the user `alice`, put another message using a sample application just as before:

```
amqsput TEST.Q QM_VERIFY_AMS
```

4. As the user bob, browse the message using a sample application via the alias queue this time:

```
amqsbcg TEST.ALIAS QM_VERIFY_AMS
```

5. As the user bob, get the message using a sample application from the local queue:

```
amqsget TEST.Q QM_VERIFY_AMS
```

Results

The output from the `amqsbcg` application shows the encrypted data that is on the queue proving that the message has been encrypted.

Linux AIX *Quick Start Guide for AMS on AIX and Linux*

Use this guide to quickly configure Advanced Message Security to provide message security on AIX and Linux. By the time you complete it, you will have created a key database to verify user identities, and defined signing/encryption policies for your queue manager.

Before you begin

You should have at least the following components installed on your system:

- Runtime
- Server
- Sample programs
- IBM Global Security Kit (GSKit)
- Advanced Message Security

Refer to the following topics for the component names on each specific platform:

- Linux IBM MQ components for Linux systems
- AIX IBM MQ components for AIX systems

1. Creating a queue manager and a queue

About this task

All the following examples use a queue named `TEST.Q` for passing messages between applications. Advanced Message Security uses interceptors to sign and encrypt messages at the point they enter the IBM MQ infrastructure through the standard IBM MQ interface. The basic setup is done in IBM MQ and is configured in the following steps.

You can use IBM MQ Explorer to create the queue manager QM_VERIFY_AMS and its local queue called `TEST.Q` by using all the default wizard settings, or you can use the commands found in *`MQ_INSTALLATION_PATH`*`/bin`. Remember that you must be a member of the `mqm` user group to run the following administrative commands.

Procedure

1. Create a queue manager

```
crtmqm QM_VERIFY_AMS
```

2. Start the queue manager

```
strmqm QM_VERIFY_AMS
```

3. Create a queue called `TEST.Q` by entering the following command into **`runmqsc`** for queue manager `QM_VERIFY_AMS`

```
DEFINE QLOCAL(TEST.Q)
```

Results

If the procedure completed successfully, the following command entered into **`runmqsc`** will display details about `TEST.Q`:

```
DISPLAY Q(TEST.Q)
```

2. Creating and authorizing users

About this task

There are two users that appear in this example: `alice`, the sender, and bob, the receiver. To use the application queue, these users need to be granted authority to use it. Also to successfully use the protection policies that we will define these users must be granted access to some system queues. For more information about the **`setmqaut`** command refer to **`setmqaut`**.

Procedure

1. Create the two users

```
useradd alice
```

```
useradd bob
```

2. Authorize the users to connect to the queue manager and to work with the queue

```
setmqaut -m QM_VERIFY_AMS -t qmgr -p alice -p bob +connect +inq
```

```
setmqaut -m QM_VERIFY_AMS -n TEST.Q -t queue -p alice +put
```

```
setmqaut -m QM_VERIFY_AMS -n TEST.Q -t queue -p bob +get
```

3. You should also allow the two users to browse the system policy queue and put messages on the error queue.

```
setmqaut -m QM_VERIFY_AMS -t queue -n SYSTEM.PROTECTION.POLICY.QUEUE -p alice -p bob +browse
```

```
setmqaut -m QM_VERIFY_AMS -t queue -n SYSTEM.PROTECTION.ERROR.QUEUE -p alice -p bob +put
```

Attention: IBM MQ optimizes performance by caching policies so that you do not have to browse records for policy details on the SYSTEM.PROTECTION.POLICY.QUEUE in all cases.

IBM MQ does not cache all the policies available. If there are high number of policies, IBM MQ caches a limited number of policies. So, if the queue manager has a low number of policies defined, there is no need to provide the browse option to the SYSTEM.PROTECTION.POLICY.QUEUE.

However, you should give browse authority to this queue, in case there is a high number of policies defined, or if you are using old clients. The SYSTEM.PROTECTION.ERROR.QUEUE is used to put error messages generated by the AMS code. The put authority against this queue is checked only when you attempt to put an error message to the queue. Your put authority against the queue is not checked when you attempt to put or get message from an AMS protected queue.

Results

User groups are now created and the required authorities granted to them. This way users who are assigned to those groups will also have permission to connect to the queue manager and to put and get from the queue.

What to do next

To verify if the steps were carried out correctly, use the `amqsput` and `amqsget` samples as described in section "8. Testing encryption" on page 604.

3. Creating key database and certificates

About this task

To encrypt the message, the interceptor requires the private key of the sending user and the public key(s) of the recipient(s). Thus, the key database of user identities mapped to public and private keys must be created. In the real system, where users and applications are dispersed over several computers, each user would have its own private keystore. Similarly, in this guide, we create key databases for `alice` and bob and share the user certificates between them.

Note: In this guide, we use sample applications written in C connecting using local bindings. If you plan to use Java applications using client bindings, you must create a JKS keystore and certificates using the **`keytool`** command, which is part of the JRE (see "Quick Start Guide for AMS with Java clients" on page 614 for more details). For all other languages, and for Java applications using local bindings, the steps in this guide are correct.

Procedure

1. Create a new key database for the user `alice`

```
mkdir /home/alice/.mqs -p
```

```
runmqakm -keydb -create -db /home/alice/.mqs/alicekey.kdb -pw passw0rd -stash
```

Note:

- It is advisable to use a strong password to secure the database.

- The **stash** parameter stores the password into the `key.sth` file, which interceptors can use to open the database.

2. Ensure the key database is readable

```
chmod +r /home/alice/.mqs/alicekey.kdb
```

3. Create a certificate identifying the user `alice` for use in encryption

```
runmqakm -cert -create -db /home/alice/.mqs/alicekey.kdb -pw passw0rd -label Alice_Cert -dn
"cn=alice,O=IBM,c=GB" -default_cert yes
```

Note:

- For the purpose of this guide, we are using self-signed certificate which can be created without using a Certificate Authority. For production systems, it is advisable not to use self-signed certificates but instead rely on certificates signed by a Certificate Authority.
- 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.

4. Now we have created the key database, we should set the ownership of it, and ensure it is unreadable by all other users.

```
chown alice /home/alice/.mqs/alicekey.kdb /home/alice/.mqs/alicekey.sth
```

```
chmod 600 /home/alice/.mqs/alicekey.kdb /home/alice/.mqs/alicekey.sth
```

5. Repeat step 1-4 for the user bob

Results

The two users `alice` and bob each now have a self-signed certificate.

4. Creating keystore.conf

About this task

You must point Advanced Message Security interceptors to the directory where the key databases and certificates are located. This is done via the `keystore.conf` file, which holds that information in plain text form. Each user must have a separate `keystore.conf` file in the `.mqs` folder. This step must be done for both `alice` and bob.

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

```
cms.keystore = dir/keystore_file
```

```
cms.certificate = certificate_label
```

Example

For this scenario, the contents of the `keystore.conf` will be as follows:

```
cms.keystore = /home/alice/.mqs/alicekey
cms.certificate = Alice_Cert
```

Note:

- The path to the keystore file must be provided with no file extension.
- There are the following keystore formats: CMS (Cryptographic Message Syntax), JKS (Java Keystore) and JCEKS (Java Cryptographic Extension Keystore). For more information, refer to "Structure of the keystore configuration file (keystore.conf) for AMS" on page 628.
- `HOME/.mqs/keystore.conf` is the default location where Advanced Message Security searches for the `keystore.conf` file. For information about how to use a non-default location for the `keystore.conf`, see "Using keystores and certificates with AMS" on page 627.

5. Sharing Certificates

About this task

Share the certificates between the two key databases so that each user can successfully identify the other. This is done by extracting each user's public certificate to a file, which is then added to the other user's key database.

Note: Take care to use the *extract* option, and not the *export* option. *Extract* gets the user's public key, whereas *export* gets both the public and private key. Using *export* by mistake would completely compromise your application, by passing on its private key.

Procedure

1. Extract the certificate identifying `alice` to an external file:

   ```
   runmqakm -cert -extract -db /home/alice/.mqs/alicekey.kdb -pw passw0rd -label Alice_Cert
   -target alice_public.arm
   ```
2. Add the certificate to `bob's` keystore:

   ```
   runmqakm -cert -add -db /home/bob/.mqs/bobkey.kdb -pw passw0rd -label Alice_Cert -file
   alice_public.arm
   ```
3. Repeat the step for bob:

   ```
   runmqakm -cert -extract -db /home/bob/.mqs/bobkey.kdb -pw passw0rd -label Bob_Cert -target
   bob_public.arm
   ```
4. Add the certificate for bob to `alice's` keystore:

   ```
   runmqakm -cert -add -db /home/alice/.mqs/alicekey.kdb -pw passw0rd -label Bob_Cert -file
   bob_public.arm
   ```

Results

The two users `alice` and bob are now able to successfully identify each other having created and shared self-signed certificates.

What to do next

Verify that a certificate is in the keystore by running the following commands which print out its details:

```
runmqakm -cert -details -db /home/bob/.mqs/bobkey.kdb -pw passw0rd -label Alice_Cert
```

```
runmqakm -cert -details -db /home/alice/.mqs/alicekey.kdb -pw passw0rd -label Bob_Cert
```

6. Defining queue policy

About this task

With the queue manager created and interceptors prepared to intercept messages and access encryption keys, we can start defining protection policies on `QM_VERIFY_AMS` using the `setmqspl` command. Refer to setmqspl for more information on this command. Each policy name must be the same as the queue name it is to be applied to.

Example

This is an example of a policy defined for the `TEST.Q` queue. In this example, messages are signed by the user `alice` using the Deprecated SHA1 algorithm, and encrypted using the 256-bit AES algorithm. `alice` is the only valid sender and bob is the only receiver of the messages on this queue:

```
setmqspl -m QM_VERIFY_AMS -p TEST.Q -s SHA1 -a "CN=alice,O=IBM,C=GB" -e AES256 -r
"CN=bob,O=IBM,C=GB"
```

Note: The DNs match exactly those specified in the respective user's certificate from the key database.

What to do next

To verify the policy you have defined, issue the following command:

```
dspmqspl -m QM_VERIFY_AMS
```

To print the policy details as a set of `setmqspl` commands, use the `-export` flag. This allows storing already defined policies:

```
dspmqspl -m QM_VERIFY_AMS -export >restore_my_policies.bat
```

7. Testing the setup

About this task

By running different programs under different users you can verify if the application has been properly configured.

Procedure

1. Change to the directory containing the samples. If MQ is installed in a non-default location, this may be in a different place.

   ```
   cd /opt/mqm/samp/bin
   ```

2. Switch user to run as user `alice`

   ```
   su alice
   ```

3. As the user `alice`, put a message using a sample application:

   ```
   ./amqsput TEST.Q QM_VERIFY_AMS
   ```

4. Type the text of the message, then press Enter.
5. Stop running as user `alice`

   ```
   exit
   ```

6. Switch user to run as user bob

   ```
   su bob
   ```

7. As the user bob, get a message using a sample application:

   ```
   ./amqsget TEST.Q QM_VERIFY_AMS
   ```

Results

If the application has been configured properly for both users, the user `alice` 's message is displayed when bob runs the getting application.

8. Testing encryption

About this task

To verify that the encryption is occurring as expected, create an alias queue which references the original queue `TEST.Q`. This alias queue will have no security policy and so no user will have the information to decrypt the message and therefore the encrypted data will be shown.

Procedure

1. Using the **`runmqsc`** command against queue manager QM_VERIFY_AMS, create an alias queue.

   ```
   DEFINE QALIAS(TEST.ALIAS) TARGET(TEST.Q)
   ```

2. Grant bob access to browse from the alias queue

   ```
   setmqaut -m QM_VERIFY_AMS -n TEST.ALIAS -t queue -p bob +browse
   ```

3. As the user `alice`, put another message using a sample application just as before:

   ```
   ./amqsput TEST.Q QM_VERIFY_AMS
   ```

4. As the user bob, browse the message using a sample application via the alias queue this time:

   ```
   ./amqsbcg TEST.ALIAS QM_VERIFY_AMS
   ```

5. As the user bob, get the message using a sample application from the local queue:

   ```
   ./amqsget TEST.Q QM_VERIFY_AMS
   ```

Results

The output from the `amqsbcg` application will show the encrypted data that is on the queue proving that the message has been encrypted.

z/OS *Example AMS configurations on z/OS*

This section provides example configurations of policies and certificates for Advanced Message Security queuing scenarios on z/OS.

See Configuring Advanced Message Security for z/OS for details on how you configure Advanced Message Security.

The examples cover the Advanced Message Security policies required, and the digital certificates that must exist relative to users and key rings. The examples assume that the users involved in the scenarios have been set up by following the instructions provided in Grant users resource permissions for Advanced Message Security.

See also server-to-server message channel interception examples.

z/OS *Local queuing of integrity-protected messages for AMS on z/OS*
This example details the Advanced Message Security policies and certificates needed to send and retrieve integrity-protected messages to and from a queue, local to the putting and getting applications.

The example queue manager and queue are:

```
BNK6        - Queue manager
FIN.XFER.Q7 - Local queue
```

These users are used:

```
WMQBNK6  - AMS task user
TELLER5  - Sending user
FINADM2  - Recipient user
```

Create the user certificates

In this example, only one user certificate is needed. This is the sending user's certificate which is needed to sign integrity-protected messages. The sending user is 'TELLER5'.

The Certificate Authority (CA) certificate is also required. The CA certificate is the certificate of the authority that issued the user's certificate. This can be a chain of certificates. If so, all certificates in the chain are required in the key ring of the Advanced Message Security task user, in this case user WMQBNK6.

A CA certificate can be created using the RACF RACDCERT command. This certificate is used to issue user certificates. For example:

```
RACDCERT CERTAUTH GENCERT SUBJECTSDN(CN('BCOCA') O('BCO') C('US'))
KEYUSAGE(CERTSIGN) WITHLABEL('BCOCA')
```

This RACDCERT command creates a CA certificate which can then be used to issue a user certificate for user 'TELLER5'. For example:

```
RACDCERT ID(TELLER5) GENCERT SUBJECTSDN(CN('Teller5') O('BCO') C('US'))
WITHLABEL('Teller5') SIGNWITH(CERTAUTH LABEL('BCOCA'))KEYUSAGE(HANDSHAKE DATAENCRYPT DOCSIGN)
```

Your installation will have procedures for choosing or creating a CA certificate, as well as procedures for issuing certificates and distributing them to relevant systems.

When exporting and importing these certificates, Advanced Message Security requires:

- The CA certificate (chain).
- The user certificate and its private key.

If you are using RACF, the RACDCERT EXPORT command can be used to export certificates to a data set, and the RACDCERT ADD command can be used to import certificates from the data set. For more information about these and other RACDCERT commands, refer to *z/OS: Security Server RACF Command Language Reference*.

The certificates in this case, are required on the z/OS system running queue manager BNK6.

When the certificates have been imported on the z/OS system running BNK6, the user certificate requires the TRUST attribute. The RACDCERT ALTER command can be used to add the TRUST attribute to the certificate. For example:

```
RACDCERT ID(TELLER5) ALTER (LABEL('Teller5')) TRUST
```

In this example, no certificate is required for the recipient user.

Connect certificates to relevant key rings

When the required certificates have been created or imported, and set as trusted, they must be connected to the appropriate user key rings on the z/OS system running BNK6. To create the key rings use the RACDCERT ADDRING commands:

```
RACDCERT ID(WMQBNK6) ADDRING(drq.ams.keyring)
```

```
RACDCERT ID(TELLER5) ADDRING(drq.ams.keyring)
```

This creates a key ring for the Advanced Message Security task user, WMQBNK6, and a key ring for the sending user, 'TELLER5'. Note that the key ring name drq.ams.keyring is mandatory, and the name is case-sensitive.

When the key rings have been created, the relevant certificates can be connected:

```
RACDCERT ID(WMQBNK6) CONNECT(CERTAUTH LABEL('BCOCA')
RING(drq.ams.keyring))
```

```
RACDCERT ID(TELLER5) CONNECT(ID(TELLER5) LABEL('Teller5')
RING(drq.ams.keyring) DEFAULT USAGE(PERSONAL))
```

The sending user certificate must be connected as DEFAULT. If the sending user has more than one certificate in its drq.ams.keyring, the default certificate is used for signing purposes.

The creation and modification of certificates is not recognized by Advanced Message Security until the queue manager is stopped and restarted, or the z/OS **MODIFY** command is used to refresh the Advanced Message Security certificate configuration. For example:

```
F BNK6AMSM,REFRESH KEYRING
```

Create the Advanced Message Security policy

In this example, integrity-protected messages are put to queue FIN.XFER.Q7 by an application running as user 'TELLER5', and retrieved from the same queue by an application running as user 'FINADM2', so only one Advanced Message Security policy is required.

Advanced Message Security policies are created using the CSQ0UTIL utility that is documented at The message security policy utility (CSQ0UTIL).

Use the CSQ0UTIL utility to run the following command:

```
setmqspl -m BNK6 -p FIN.XFER.Q7 -s MD5 -a CN=Teller5,O=BCO,C=US
```

In this policy, the queue manager is identified as BNK6. The policy name and associated queue is FIN.XFER.Q7. The algorithm that is used to generate the sender's signature is MD5, and the distinguished name (DN) of the sending user is 'CN=Teller5,O=BCO,C=US'.

After defining the policy, either restart the BNK6 queue manager, or use the z/OS **MODIFY** command to refresh the Advanced Message Security policy configuration. For example:

```
F BNK6AMSM,REFRESH POLICY
```

z/OS *Local queuing of privacy-protected messages for AMS on z/OS*

This example details the Advanced Message Security policies and certificates needed to send and retrieve privacy-protected messages to and from a queue, local to the putting and getting applications. Privacy-protected messages are both signed and encrypted.

The example queue manager and local queue are as follows:

```
BNK6        - Queue manager
FIN.XFER.Q8 - Local queue
```

These users are used:

```
WMQBNK6  - AMS task user
TELLER5  - Sending user
FINADM2  - Recipient user
```

The steps to configure this scenario are:

Create the user certificates

In this example, two user certificates are required. These are the sending user's certificate which is needed to sign messages, and the recipient user's certificate which is needed to encrypt and decrypt the message data. The sending user is 'TELLER5' and the recipient user is 'FINADM2'.

The Certificate Authority (CA) certificate is also required. The CA certificate is the certificate of the authority that issued the user's certificate. This can be a chain of certificates. If so, all certificates in the chain are required in the key ring of the Advanced Message Security task user, in this case user WMQBNK6.

A CA certificate can be created using the RACF RACDCERT command. This certificate is used to issue user certificates. For example:

```
RACDCERT CERTAUTH GENCERT SUBJECTSDN(CN('BCOCA') O('BCO') C('US'))
KEYUSAGE(CERTSIGN) WITHLABEL('BCOCA')
```

This RACDCERT command creates a CA certificate which can then be used to issue user certificates for users 'TELLER5' and 'FINADM2'. For example:

```
RACDCERT ID(TELLER5) GENCERT SUBJECTSDN(CN('Teller5') O('BCO') C('US'))
WITHLABEL('Teller5') SIGNWITH(CERTAUTH LABEL('BCOCA'))KEYUSAGE(HANDSHAKE DATAENCRYPT DOCSIGN)
```

```
RACDCERT ID(FINADM2) GENCERT SUBJECTSDN(CN('FinAdm2') O('BCO') C('US'))
WITHLABEL('FinAdm2') SIGNWITH(CERTAUTH LABEL('BCOCA'))KEYUSAGE(HANDSHAKE DATAENCRYPT DOCSIGN)
```

Your installation will have procedures for choosing or creating a CA certificate, as well as procedures for issuing certificates and distributing them to relevant systems.

When exporting and importing these certificates, Advanced Message Security requires:

- The CA certificate (chain).
- The sending user certificate and its private key.
- The recipient user certificate and its private key.

If you are using RACF, the RACDCERT EXPORT command can be used to export certificates to a data set, and the RACDCERT ADD command can be used to import certificates from the data set. For more information about these and other RACDCERT commands, refer to RACDCERT (Manage RACF digital certificates) in the *z/OS: Security Server RACF Command Language Reference*.

The certificates in this case are required on the z/OS system running queue manager BNK6.

When the certificates have been imported on the z/OS system running BNK6, the user certificates require the TRUST attribute. The RACDCERT ALTER command can be used to add the TRUST attribute to the certificate. For example:

```
RACDCERT ID(TELLER5) ALTER (LABEL('Teller5')) TRUST
```

```
RACDCERT ID(FINADM2) ALTER (LABEL('FinAdm2')) TRUST
```

Connect certificates to relevant key rings

When the required certificates have been created or imported, and set as trusted, they must be connected to the appropriate user key rings on the z/OS system running BNK6. To create the key rings use the RACDCERT ADDRING command:

```
RACDCERT ID(WMQBNK6) ADDRING(drq.ams.keyring)
```

```
RACDCERT ID(TELLER5) ADDRING(drq.ams.keyring)
```

```
RACDCERT ID(FINADM2) ADDRING(drq.ams.keyring)
```

This creates a key ring for the Advanced Message Security task user and key rings for the sending and recipient users. Note that the key ring name `drq.ams.keyring` is mandatory, and the name is case-sensitive.

When the key rings have been created, the relevant certificates can be connected.

```
RACDCERT ID(WMQBNK6) CONNECT(CERTAUTH LABEL('BCOCA')
RING(drq.ams.keyring))
```

```
RACDCERT ID(WMQBNK6) CONNECT(ID(FINADM2) LABEL('FinAdm2')
RING(drq.ams.keyring) USAGE(SITE))
```

```
RACDCERT ID(TELLER5) CONNECT(ID(TELLER5) LABEL('Teller5')
RING(drq.ams.keyring) DEFAULT USAGE(PERSONAL))
```

```
RACDCERT ID(FINADM2) CONNECT(ID(FINADM2) LABEL('FinAdm2')
RING(drq.ams.keyring) DEFAULT USAGE(PERSONAL))
```

The sending and recipient user certificates must be connected as DEFAULT. If either user has more than one certificate in its drq.ams.keyring, the default certificate is used for signing and decryption purposes.

The recipient user's certificate must also be connected to the Advanced Message Security task user's key ring with USAGE(SITE). This is because the Advanced Message Security task needs the recipient's public key when encrypting the message data. The USAGE(SITE) prevents the private key from being accessible in the key ring.

The creation and modification of certificates is not recognized by Advanced Message Security until the queue manager is stopped and restarted, or the z/OS **MODIFY** command is used to refresh the Advanced Message Security certificate configuration. For example:

```
F BNK6AMSM,REFRESH KEYRING
```

Create the Advanced Message Security policy

In this example, privacy-protected messages are put to queue FIN.XFER.Q8 by an application running as user 'TELLER5', and retrieved from the same queue by an application running as user 'FINADM2', so only one Advanced Message Security policy is required.

Advanced Message Security policies are created using the CSQ0UTIL utility that is documented at The message security policy utility (CSQ0UTIL).

Use the CSQ0UTIL utility to run the following command:

```
setmqspl -m BNK6 -p FIN.XFER.Q8 -s SHA1 -e 3DES -a CN=Teller5,O=BCO,C=US -r
CN=FinAdm2,O=BCO,C=US
```

In this policy, the queue manager is identified as BNK6. The policy name and associated queue is FIN.XFER.Q8. The algorithm that is used to generate the sender's signature is Deprecated SHA1, and the distinguished name (DN) of the sending user is 'CN=Teller5,O=BCO,C=US', and the recipient user is 'CN=FinAdm2,O=BCO,C=US'. The algorithm that is used to encrypt the message data is Deprecated 3DES.

After defining the policy, either restart the BNK6 queue manager, or use the z/OS **MODIFY** command to refresh the Advanced Message Security policy configuration. For example:

```
F BNK6AMSM,REFRESH POLICY
```

z/OS *Remote queuing of integrity-protected messages for AMS on z/OS*
This example details the Advanced Message Security policies and certificates needed to send and retrieve integrity-protected messages to and from queues managed by two different queue managers. The two queue managers can be running on the same z/OS system, or on different z/OS systems, or one queue manager can be on a distributed system running Advanced Message Security.

The example queue managers and queues are:

```
BNK6        - Sending queue manager
BNK7        - Recipient queue manager
FIN.XFER.Q7 - Remote queue on BNK6
FIN.RCPT.Q7 - Local queue on BNK7
```

Note: For this example, BNK6 and BNK7 are queue managers running on different z/OS systems.

These users are used:

```
WMQBNK6  - AMS task user on BNK6
WMQBNK7  - AMStask user on BNK7
TELLER5  - Sending user on BNK6
FINADM2  - Recipient user on BNK7
```

The steps to configure this scenario are as follows:

Create the user certificates

In this example, only one user certificate is needed. This is the sending user's certificate which is needed to sign integrity-protected message. The sending user is 'TELLER5'.

The Certificate Authority (CA) certificate is also required. The CA certificate is the certificate of the authority that issued the user's certificate. This can be a chain of certificates. If so, all certificates in the chain are required in the key ring of the Advanced Message Security task user, in this case user WMQBNK7.

A CA certificate can be created using the RACF RACDCERT command. This certificate is used to issue user certificates. For example:

```
RACDCERT CERTAUTH GENCERT SUBJECTSDN(CN('BCOCA') O('BCO') C('US'))
KEYUSAGE(CERTSIGN) WITHLABEL('BCOCA')
```

This RACDCERT command creates a CA certificate which can then be used to issue user certificate for user 'TELLER5'. For example:

```
RACDCERT ID(TELLER5) GENCERT SUBJECTSDN(CN('Teller5') O('BCO') C('US'))
WITHLABEL('Teller5') SIGNWITH(CERTAUTH LABEL('BCOCA'))KEYUSAGE(HANDSHAKE DATAENCRYPT DOCSIGN)
```

Your installation will have procedures for choosing or creating a CA certificate, as well as procedures for issuing certificates and distributing them to relevant systems.

When exporting and importing these certificates, Advanced Message Security require:

- The CA certificate (chain).
- The sending user certificate and its private key.

If you are using RACF, the RACDCERT EXPORT command can be used to export certificates to a data set, and the RACDCERT ADD command can be used to import certificates from the data set. For more information about these and other RACDCERT commands, refer to RACDCERT (Manage RACF digital certificates) in the *z/OS: Security Server RACF Command Language Reference*.

The certificates in this case, are required on the z/OS system running queue manager BNK6 and BNK7.

In this example, the sending certificate must be imported on the z/OS system running BNK6, and the CA certificate must be imported on the z/OS system running BNK7. When the certificates have been imported, the user certificate requires the TRUST attribute. The RACDCERT ALTER command can be used to add the TRUST attribute to the certificate. For example, on BNK6:

```
RACDCERT ID(TELLER5) ALTER (LABEL('Teller5')) TRUST
```

Connect certificates to relevant key rings

When the required certificates have been created or imported, and set as trusted, they must be connected to the appropriate user key rings on the z/OS system running BNK6 and BNK7.

To create the key rings use the RACDCERT ADDRING command, on BNK6:

```
RACDCERT ID(TELLER5) ADDRING(drq.ams.keyring)
```

This creates a key ring for the sending user on BNK6. Note that the key ring name drq.ams.keyring is mandatory, and the name is case-sensitive.

On BNK7:

```
RACDCERT ID(WMQBNK7) ADDRING(drq.ams.keyring)
```

This creates a key ring for the Advanced Message Security task user on BNK7. No user key ring is required for 'TELLER5' on BNK7.

When the key rings have been created, the relevant certificates can be connected.

On BNK6:

```
RACDCERT ID(TELLER5) CONNECT(ID(TELLER5) LABEL('Teller5')
RING(drq.ams.keyring) DEFAULT USAGE(PERSONAL))
```

On BNK7:

```
RACDCERT ID(WMQBNK7) CONNECT(CERTAUTH LABEL('BCOCA')
RING(drq.ams.keyring))
```

The sending user certificate must be connected as DEFAULT. If the sending user has more than one certificate in its drq.ams.keyring, the default certificate is used for signing purposes.

The creation and modification of certificates is not recognized by Advanced Message Security until the queue manager is stopped and restarted, or the z/OS **`MODIFY`** command is used to refresh the Advanced Message Security certificate configuration. For example:

On BNK6:

```
F BNK6AMSM,REFRESH,KEYRING
```

On BNK7:

```
F BNK7AMSM,REFRESH,KEYRING
```

Create the Advanced Message Security policies

In this example, integrity-protected messages are put to remote queue FIN.XFER.Q7 on BNK6 by an application running as user 'TELLER5', and retrieved from local queue FIN.RCPT.Q7 on BNK7 by an application running as user 'FINADM2', so two Advanced Message Security policies are required.

Advanced Message Security policies are created using the CSQ0UTIL utility that is documented at The message security policy utility (CSQ0UTIL).

Use the CSQ0UTIL utility to run the following command to define an integrity policy for the remote queue on BNK6:

```
setmqspl -m BNK6 -p FIN.XFER.Q7 -s MD5 -a CN=Teller5,O=BCO,C=US
```

In this policy, the queue manager is identified as BNK6. The policy name and associated queue is FIN.XFER.Q7. The algorithm that is used to generate the sender's signature is MD5, and the distinguished name (DN) of the sending user is 'CN=Teller5,O=BCO,C=US'.

Also, use the CSQ0UTIL utility to run the following command to define an integrity policy for the local queue on BNK7:

```
setmqspl -m BNK7 -p FIN.RCPT.Q7 -s MD5 -a CN=Teller5,O=BCO,C=US
```

In this policy, the queue manager is identified as BNK7. The policy name and associated queue is FIN.RCPT.Q7. The algorithm expected for the sender's signature is MD5, and the distinguished name (DN) of the sending user is expected to be 'CN=Teller5,O=BCO,C=US'.

After defining the two policies, either restart the BNK6 and BNK7 queue managers, or use the z/OS **MODIFY** command to refresh the Advanced Message Security policy configurations. For example:

On BNK6:

```
F BNK6AMSM,REFRESH,POLICY
```

On BNK7:

```
F BNK7AMSM,REFRESH,POLICY
```

z/OS *Remote queuing of privacy-protected messages for AMS on z/OS*
This example details the Advanced Message Security policies and certificates needed to send and retrieve privacy-protected messages to and from queues managed by two different queue managers. The two queue managers can be running on the same z/OS system, or on different z/OS systems, or one queue manager can be on a distributed system running Advanced Message Security.

The example queue managers and queues are:

```
BNK6        - Sending queue manager
BNK7        - Recipient queue manager
FIN.XFER.Q7 - Remote queue on BNK6
FIN.RCPT.Q7 - Local queue on BNK7
```

Note: For this example, BNK6 and BNK7 are queue managers running on different z/OS systems of the same name.

These users are used:

```
WMQBNK6  - AMS task user on BNK6
WMQBNK7  - AMS task user on BNK7
```

```
TELLER5  - Sending user on BNK6
FINADM2  - Recipient user on BNK7
```

The steps to configure this scenario are as follows:

Create the user certificates

In this example, two user certificates are required. These are the sending user's certificate which is needed to sign messages, and the recipient user's certificate which is needed to encrypt and decrypt the message data. The sending user is 'TELLER5' and the recipient user is 'FINADM2'.

The Certificate Authority (CA) certificate is also required. The CA certificate is the certificate of the authority that issued the user's certificate. This can be a chain of certificates. If so, all certificates in the chain are required in the key ring of the Advanced Message Security task user, in this case user WMQBNK7.

A CA certificate can be created using the RACF RACDCERT command. This certificate is used to issue user certificates. For example:

```
RACDCERT CERTAUTH GENCERT SUBJECTSDN(CN('BCOCA') O('BCO') C('US'))
KEYUSAGE(CERTSIGN) WITHLABEL('BCOCA')
```

This RACDCERT command creates a CA certificate which can then be used to issue user certificates for users 'TELLER5' and 'FINADM2'. For example:

```
RACDCERT ID(TELLER5) GENCERT SUBJECTSDN(CN('Teller5') O('BCO') C('US'))
WITHLABEL('Teller5') SIGNWITH(CERTAUTH LABEL('BCOCA'))KEYUSAGE(HANDSHAKE DATAENCRYPT DOCSIGN)
```

```
RACDCERT ID(FINADM2) GENCERT SUBJECTSDN(CN('FinAdm2') O('BCO') C('US'))
WITHLABEL('FinAdm2') SIGNWITH(CERTAUTH LABEL('BCOCA'))KEYUSAGE(HANDSHAKE DATAENCRYPT DOCSIGN)
```

Your installation will have procedures for choosing or creating a CA certificate, as well as procedures for issuing certificates and distributing them to relevant systems.

When exporting and importing these certificates, Advanced Message Security requires:

- The CA certificate (chain).
- The sending user certificate and its private key.
- The recipient user certificate and its private key.

If you are using RACF, the RACDCERT EXPORT command can be used to export certificates to a data set, and the RACDCERT ADD command can be used to import certificates from the data set.

For more information about these and other RACDCERT commands, see RACDCERT (Manage RACF digital certificates) in the *z/OS: Security Server RACF Command Language Reference*.

The certificates in this case, are required on the z/OS system running queue manager BNK6 and BNK7.

In this example, the sending and recipient certificates must be imported on the z/OS system running BNK6, and the CA and recipient certificates must be imported on the z/OS system running BNK7. When the certificates have been imported, the user certificates require the TRUST attribute. The RACDCERT ALTER command can be used to add the TRUST attribute to the certificate. For example:

On BNK6:

```
RACDCERT ID(TELLER5) ALTER (LABEL('Teller5')) TRUST
```

```
RACDCERT ID(FINADM2) ALTER (LABEL('FinAdm2')) TRUST
```

On BNK7:

```
RACDCERT ID(FINADM2) ALTER (LABEL('FinAdm2')) TRUST
```

Connect certificates to relevant key rings

When the required certificates have been created or imported, and set as trusted, they must be connected to the appropriate user key rings on the z/OS systems running BNK6 and BNK7.

To create the key rings use the RACDCERT ADDRING command:

On BNK6:

```
RACDCERT ID(WMQBNK6) ADDRING(drq.ams.keyring)
```

```
RACDCERT ID(TELLER5) ADDRING(drq.ams.keyring)
```

This creates a key ring for the Advanced Message Security task user and a key ring for the sending user on BNK6. Note that the key ring name drq.ams.keyring is mandatory, and the name is case-sensitive.

On BNK7:

```
RACDCERT ID(WMQBNK7) ADDRING(drq.ams.keyring)
```

```
RACDCERT ID(FINADM2) ADDRING(drq.ams.keyring)
```

This creates a key ring for the Advanced Message Security task user and a key ring for the recipient user on BNK7.

When the key rings have been created, the relevant certificates can be connected.

On BNK6:

```
RACDCERT ID(WMQBNK6) CONNECT(ID(FINADM2) LABEL('FinAdm2')
RING(drq.ams.keyring) USAGE(SITE))
```

```
RACDCERT ID(TELLER5) CONNECT(ID(TELLER5) LABEL('Teller5')
RING(drq.ams.keyring) DEFAULT USAGE(PERSONAL))
```

On BNK7:

```
RACDCERT ID(WMQBNK7) CONNECT(CERTAUTH LABEL('BCOCA')
RING(drq.ams.keyring))
```

```
RACDCERT ID(FINADM2) CONNECT(ID(FINADM2) LABEL('FinAdm2')
RING(drq.ams.keyring) DEFAULT USAGE(PERSONAL))
```

The sending and recipient user certificates must be connected as DEFAULT. If either user has more than one certificate in its drq.ams.keyring, the default certificate is used for signing and encryption/decryption purposes.

On BNK6, the recipient user's certificate must also be connected to the Advanced Message Security task user's key ring with USAGE(SITE). This is because the Advanced Message Security task needs the recipient's public key when encrypting the message data. The USAGE(SITE) prevents the private key from being accessible in the key ring.

The creation and modification of certificates is not recognized by Advanced Message Security until the queue manager is stopped and restarted, or the z/OS **`MODIFY`** command is used to refresh the Advanced Message Security certificate configuration. For example:

On BNK6:

```
F BNK6AMSM,REFRESH,KEYRING
```

On BNK7:

```
F BNK7AMSM,REFRESH,KEYRING
```

Create the Advanced Message Security policies

In this example, privacy-protected messages are put to remote queue FIN.XFER.Q7 on BNK6 by an application running as user 'TELLER5', and retrieved from local queue FIN.RCPT.Q7 on BNK7 by an application running as user 'FINADM2', so two Advanced Message Security policies are required.

Advanced Message Security policies are created using the CSQ0UTIL utility that is documented at The message security policy utility (CSQ0UTIL).

Use the CSQ0UTIL utility to run the following command to define a privacy policy for the remote queue on BNK6:

```
setmqspl -m BNK6 -p FIN.XFER.Q7 -s SHA1 -e 3DES -a CN=Teller5,O=BCO,C=US -r
CN=FinAdm2,O=BCO,C=US
```

In this policy, the queue manager is identified as BNK6. The policy name and associated queue is FIN.XFER.Q7. The algorithm that is used to generate the sender's signature is Deprecated SHA1, the distinguished name (DN) of the sending user is 'CN=Teller5,O=BCO,C=US', and the recipient user is 'CN=FinAdm2,O=BCO,C=US'. The algorithm that is used to encrypt the message data is Deprecated 3DES.

Also, use the CSQ0UTIL utility to run the following command to define a privacy policy for the local queue on BNK7:

```
setmqspl -m BNK7 -p FIN.RCPT.Q7 -s SHA1 -e 3DES -a CN=Teller5,O=BCO,C=US -r
CN=FinAdm2,O=BCO,C=US
```

In this policy, the queue manager is identified as BNK7. The policy name and associated queue is FIN.RCPT.Q7. The algorithm expected for the sender's signature is Deprecated SHA1, the distinguished name (DN) of the sending user is expected to be 'CN=Teller5,O=BCO,C=US', and the recipient user is 'CN=FinAdm2,O=BCO,C=US'. The algorithm that is used to decrypt the message data is Deprecated 3DES.

After defining the two policies, either restart the BNK6 and BNK7 queue managers, or use the z/OS **MODIFY** command to refresh the Advanced Message Security policy configuration. For example:

On BNK6:

```
F BNK6AMSM,REFRESH,POLICY
```

On BNK7:

```
F BNK7AMSM,REFRESH,POLICY
```

Quick Start Guide for AMS with Java clients

Use this guide to quickly configure Advanced Message Security to provide message security for Java applications connecting using client bindings. By the time you complete it, you will have created a keystore to verify user identities, and defined signing/encryption policies for your queue manager.

Before you begin

Ensure you have the appropriate components installed as described in "Quick Start Guide for AMS on Windows platforms" on page 592 or "Quick Start Guide for AMS on AIX and Linux" on page 598.

1. Creating a queue manager and a queue

About this task

All the following examples use a queue named `TEST.Q` for passing messages between applications. Advanced Message Security uses interceptors to sign and encrypt messages at the point they enter the

IBM MQ infrastructure through the standard IBM MQ interface. The basic setup is done in IBM MQ and is configured in the following steps.

Procedure

1. Create a queue manager

   ```
   crtmqm QM_VERIFY_AMS
   ```

2. Start the queue manager

   ```
   strmqm QM_VERIFY_AMS
   ```

3. Create and start a listener by entering the following commands into **runmqsc** for queue manager QM_VERIFY_AMS

   ```
   DEFINE LISTENER(AMS.LSTR) TRPTYPE(TCP) PORT(1414) CONTROL(QMGR)
   ```

   ```
   START LISTENER(AMS.LSTR)
   ```

4. Create a channel for our applications to connect in through by entering the following command into **runmqsc** for queue manager QM_VERIFY_AMS

   ```
   DEFINE CHANNEL(AMS.SVRCONN) CHLTYPE(SVRCONN)
   ```

5. Create a queue called TEST.Q by entering the following command into **runmqsc** for queue manager QM_VERIFY_AMS

   ```
   DEFINE QLOCAL(TEST.Q)
   ```

Results

If the procedure completed successfully, the following command entered into **runmqsc** displays details about TEST.Q:

```
DISPLAY Q(TEST.Q)
```

2. Creating and authorizing users

About this task

There are two users that appear in this scenario: alice, the sender, and bob, the receiver. To use the application queue, these users need to be granted authority to use it. Also to successfully use the protection policies defined in this scenario, these users must be granted access to some system queues. For more information about the **setmqaut** command refer to **setmqaut**.

Procedure

1. Create the two users as described in the **Quick Start Guide** (Windows or AIX and Linux) for your platform.
2. Authorize the users to connect to the queue manager and to work with the queue

   ```
   setmqaut -m QM_VERIFY_AMS -t qmgr -p alice -p bob +connect +inq
   ```

```
setmqaut -m QM_VERIFY_AMS -n TEST.Q -t queue -p alice +put
```

```
setmqaut -m QM_VERIFY_AMS -n TEST.Q -t queue -p bob +get +inq +browse
```

3. You should also allow the two users to browse the system policy queue and put messages on the error queue.

   ```
   setmqaut -m QM_VERIFY_AMS -t queue -n SYSTEM.PROTECTION.POLICY.QUEUE -p alice -p bob +browse
   ```

   ```
   setmqaut -m QM_VERIFY_AMS -t queue -n SYSTEM.PROTECTION.ERROR.QUEUE -p alice -p bob +put
   ```

   Attention: IBM MQ optimizes performance by caching policies so that you do not have to browse records for policy details on the SYSTEM.PROTECTION.POLICY.QUEUE in all cases.

   IBM MQ does not cache all the policies available. If there are high number of policies, IBM MQ caches a limited number of policies. So, if the queue manager has a low number of policies defined, there is no need to provide the browse option to the SYSTEM.PROTECTION.POLICY.QUEUE.

   However, you should give browse authority to this queue, in case there is a high number of policies defined, or if you are using old clients. The SYSTEM.PROTECTION.ERROR.QUEUE is used to put error messages generated by the AMS code. The put authority against this queue is checked only when you attempt to put an error message to the queue. Your put authority against the queue is not checked when you attempt to put or get message from an AMS protected queue.

Results

Users are now created and the required authorities granted to them.

What to do next

To verify if the steps were carried out correctly, use the `JmsProducer` and `JmsConsumer` samples as described in section "7. Testing the setup" on page 619.

3. Creating key database and certificates

About this task

To encrypt the message to interceptor requires the public key of the sending users. Thus, the key database of user identities mapped to public and private keys must be created. In the real system, where users and applications are dispersed over several computer, each user would have its own private keystore. Similarly, in this guide, we create key databases for `alice` and bob and share the user certificates between them.

Note: In this guide, we use sample applications written in Java connecting using client bindings. If you plan to use Java applications using local bindings or C applications, you must create a CMS keystore and certificates using the **runmqakm** command. For more information, see "Quick Start Guide for AMS on Windows platforms" on page 592 and "Quick Start Guide for AMS on AIX and Linux" on page 598.

Procedure

1. Create a directory in which to create your keystore, for example `/home/alice/.mqs`. You might wish to create it in the same directory as used by the Quick Start Guide for your platform. For more information, see "Quick Start Guide for AMS on Windows platforms" on page 592 and "Quick Start Guide for AMS on AIX and Linux" on page 598.

   Note: This directory is referred to as *keystore-dir* in the following steps

2. Create a new keystore and certificate identifying the user `alice` for use in encryption

   Note: The **`keytool`** command is part of the JRE.

   ```
   keytool -genkey -alias Alice_Java_Cert -keyalg RSA -keystore keystore-dir/keystore.jks
   -storepass passw0rd
   -dname "CN=alice, O=IBM, C=GB" -keypass passw0rd
   ```

   Note:

   - If your *keystore-dir* contains spaces, you must put quotes round the full name of your keystore
   - It is advisable to use a strong password to secure the keystore.
   - For the purpose of this guide, we are using self-signed certificate which can be created without using a Certificate Authority. For production systems, it is advisable not to use self-signed certificates but instead rely on certificates signed by a Certificate Authority.
   - The **`alias`** parameter specifies the name for the certificate, which interceptors will look up to receive necessary information.
   - The **dname** parameter specifies the details of the **Distinguished Name** (DN), which must be unique for each user.
3. On AIX and Linux, ensure the keystore is readable

   ```
   chmod +r keystore-dir/keystore.jks
   ```
4. Repeat step1-4 for the user bob

Results

The two users `alice` and bob each now have a self-signed certificate.

4. Creating keystore.conf

About this task

You must point 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. This step should be done for both `alice` and bob.

Example

For this scenario, the contents of the `keystore.conf` for `alice` are as follows:

```
JKS.keystore = keystore-dir/keystore
JKS.certificate = Alice_Java_Cert
JKS.encrypted = no
JKS.keystore_pass = passw0rd
JKS.key_pass = passw0rd
JKS.provider = IBMJCE
```

For this scenario, the contents of the `keystore.conf` for bob are as follows:

```
JKS.keystore = keystore-dir/keystore
JKS.certificate = Bob_Java_Cert
JKS.encrypted = no
JKS.keystore_pass = passw0rd
JKS.key_pass = passw0rd
JKS.provider = IBMJCE
```

Note:

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

- If you already have a `keystore.conf` file because you have followed the instructions in the Quick Start Guide (Windows or AIX and Linux), you can edit the existing file to add these lines.
- For more information, see "Structure of the keystore configuration file (keystore.conf) for AMS" on page 628.

5. Sharing certificates

About this task

Share the certificates between the two keystores so that each user can successfully identify the other. This is done by extracting each user's certificate and importing it into the other user's keystore.

Important: The terms *extract* and *export* are used differently by different certificate management commands.

- The IBM Global Security Kit (GSKit) **runmqakm** command uses the term *extract* to refer to the process of copying only the public part of a certificate from a keystore, and the term *export* to refer to the process of copying certificates and their associated public and private keys from one keystore to another.
- The Java **keytool** command, V 9.4.0 V 9.4.0 and the IBM MQ **runmqktool** command, use the term *export* to refer to the process of copying only the public part of a certificate from a keystore.

This distinction is important since using *export* incorrectly can compromise your application by exposing its private key. Because the distinction is so important, the IBM MQ documentation uses these terms consistently. For these reasons, the following procedure refers to *extracting* certificates by using the `exportcert` option in the **keytool** command.

Procedure

1. Extract the certificate identifying `alice`.

   ```
   keytool -exportcert -keystore alice-keystore-dir/keystore.jks -storepass passw0rd
   -alias Alice_Java_Cert -file alice-keystore-dir/Alice_Java_Cert.cer
   ```

2. Import the certificate identifying `alice` into the keystore that bob will use. When prompted indicate that you will trust this certificate.

   ```
   keytool -importcert -file alice-keystore-dir/Alice_Java_Cert.cer -alias Alice_Java_Cert
   -keystore bob-keystore-dir/keystore.jks -storepass passw0rd
   ```

3. Repeat the steps for bob

Results

The two users `alice` and bob are now able to successfully identify each other having created and shared self-signed certificates.

What to do next

Verify that a certificate is in the keystore by running the following commands which print out its details:

```
keytool -list -keystore bob-keystore-dir/keystore.jks -storepass passw0rd -alias Alice_Java_Cert
```

```
keytool -list -keystore alice-keystore-dir/keystore.jks -storepass passw0rd -alias Bob_Java_Cert
```

6. Defining queue policy

About this task

With the queue manager created and interceptors prepared to intercept messages and access encryption keys, we can start defining protection policies on QM_VERIFY_AMS using the `setmqspl` command. Refer to setmqspl for more information on this command. Each policy name must be the same as the queue name it is to be applied to.

Example

This is an example of a policy defined on the `TEST.Q` queue, signed by the user `alice` using the Deprecated SHA1 algorithm, and encrypted using the 256-bit AES algorithm for the user bob:

```
setmqspl -m QM_VERIFY_AMS -p TEST.Q -s SHA1 -a "CN=alice,O=IBM,C=GB" -e AES256 -r
"CN=bob,O=IBM,C=GB"
```

Note: The DNs match exactly those specified in the respective user's certificate from the key database.

What to do next

To verify the policy you have defined, issue the following command:

```
dspmqspl -m QM_VERIFY_AMS
```

To print the policy details as a set of `setmqspl` commands, the `-export` flag. This allows storing already defined policies:

```
dspmqspl -m QM_VERIFY_AMS -export >restore_my_policies.bat
```

7. Testing the setup

Before you begin

Ensure the version of Java you are using has the unrestricted JCE policy files installed.

Note: The version of Java supplied in the IBM MQ installation already has these policy files. It can be found in `*MQ_INSTALLATION_PATH*/java/bin`.

About this task

By running different programs under different users you can verify if the application has been properly configured. For more information about running programs under different users, see "Quick Start Guide for AMS on Windows platforms" on page 592 and "Quick Start Guide for AMS on AIX and Linux" on page 598.

Procedure

1. To run these JMS sample applications, use the CLASSPATH setting for your platform as shown in Environment variables used by IBM MQ classes for JMS to ensure the samples directory is included.
2. As the user `alice`, put a message using a sample application, connecting as a client:

   ```
   java JmsProducer -m QM_VERIFY_AMS -d TEST.Q -h localhost -p 1414 -l AMS.SVRCONN
   ```

3. As the user bob, get a message using a sample application, connecting as a client:

   ```
   java JmsConsumer -m QM_VERIFY_AMS -d TEST.Q -h localhost -p 1414 -l AMS.SVRCONN
   ```

Results

If the application has been configured properly for both users, the user `alice` 's message is displayed when bob runs the getting application.

Protecting remote queues on AMS

To fully protect remote queues, policies must be set on the remote queue and local queue to which messages are transmitted.

When a message is put into a remote queue, Advanced Message Security 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 the IBM MQ to handle it. After Advanced Message Security has processed the message put into a remote queue, IBM 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, Advanced Message Security tries to decode the message according to the policy set on the local queue. For the 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.

If for any reason both policies cannot be set at the same time, a staged roll-out support is provided. The policy can be set on a local queue with toleration flag on, which indicates that a policy associated with a queue can be ignored when an attempt to retrieve a message from the queue involves a message that does not have the security policy set. In this case, GET will try to decrypt the message, but will allow non-encrypted messages to be delivered. This way policies on remote queues can be set after the local queues has been protected (and tested).

Remember: Remove the toleration flag once the Advanced Message Security roll-out has been completed.

Related reference

setmqspl (set security policy)

Routing protected messages with AMS using IBM Integration Bus

Advanced Message Security can protect messages in an infrastructure where IBM Integration Bus, or WebSphere Message Broker 8.0.0.1 (or later) is installed. You should understand the nature of both products before applying security in the IBM Integration Bus environment.

About this task

Advanced Message Security provides end-to-end security of the message payload. This means that only the parties specified as the valid senders and recipients of a message are capable of producing or receiving it. This implies that in order to secure messages flowing through IBM Integration Bus, you can either allow IBM Integration Bus to process messages without knowing their content (Scenario 1) or make it an authorized user able to receive and send messages (Scenario 2).

Scenario 1 - Integration Bus cannot see message content

Before you begin

You should have your IBM Integration Bus connected to an existing queue manager. Replace *QMgrName* with this existing queue manager name in the commands that follow.

About this task

In this scenario, Alice puts a protected message into an input queue `QIN`. Based on the message property `routeTo`, the message is routed either to *bob's* (`QBOB`), [1] (`QCECIL`), or the default (`QDEF`) queue. The routing is possible because Advanced Message Security protects only the message payload and not its headers and properties which remain unprotected and can be read by IBM Integration Bus. Advanced

[1] cecil's

Message Security is used only by *alice*, *bob* and *cecil*. It is not necessary to install or configure it for the IBM Integration Bus.

IBM Integration Bus receives the protected message from the unprotected alias queue in order to avoid any attempt to decrypt the message. If it were to use the protected queue directly, the message would be put onto the DEAD LETTER queue as impossible to decrypt. The message is routed by IBM Integration Bus and arrives on the target queue unchanged. Therefore it is still signed by the original author (both *bob* and *cecil* only accept messages sent by *alice*) and protected as before (only *bob* and *cecil* can read it). IBM Integration Bus puts the routed message to an unprotected alias. The recipients retrieve the message from a protected output queue where AMS will transparently decrypt the message.

Procedure

1. Configure *alice*, *bob* and *cecil* to use Advanced Message Security as described in the **Quick Start Guide** (Windows or AIX).

   Ensure the following steps are completed:

   - Creating and authorizing users
   - Creating Key Database and Certificates
   - Creating keystore.conf
2. Provide *alice's* certificate to *bob* and *cecil*, so *alice* can be identified by them when checking digital signatures on messages.

   Do this by extracting the certificate identifying *alice* to an external file, then adding the extracted certificate to *bob's* and *cecil's* keystores. It is important that you use the method described in **Task 5. Sharing Certificates** in the **Quick Start Guide** (Windows or AIX).
3. Provide *bob* and *cecil's* certificates to *alice*, so *alice* can send messages encrypted for *bob* and *cecil*.

   Do this using the method specified in the previous step.
4. On your queue manager, define local queues called `QIN`, `QBOB`, `QCECIL` and `QDEF`.

   ```
   DEFINE QLOCAL(QIN)
   ```
5. Set up the security policy for the `QIN` queue to an eligible configuration. Use the identical setup for the `QBOB`, `QCECIL` and `QDEF` queues.

   ```
   setmqspl -m QMgrName -p QIN -s SHA1 -a "CN=alice,O=IBM,C=GB"
   -e AES256 -r "CN=bob,O=IBM,C=GB" -r "CN=cecil,O=IBM,C=GB"
   ```

   This scenario assumes the security policy where *alice* is the only authorized sender and *bob* and *cecil* are the recipients.
6. Define alias queues `AIN`, `ABOB` and `ACECIL` referencing local queues `QIN`, `QBOB` and `QCECIL` respectively.

   ```
   DEFINE QALIAS(AIN) TARGET(QIN)
   ```
7. Verify that the security configuration for the aliases specified in the previous step is not present; otherwise set its policy to NONE.

   ```
   dspmqspl -m QMgrName -p AIN
   ```
8. In IBM Integration Bus create a message flow to route the messages arriving on the `AIN` alias queue to the `BOB`, `CECIL`, or `DEF` node depending on the `routeTo` property of the message. To do so:

   a) Create an `MQInput` node called `IN` and assign the `AIN` alias as its queue name.

   b) Create `MQOutput` nodes called `BOB`, `CECIL` and `DEF`, and assign alias queues `ABOB`, `ACECIL` and `ADEF` as their respective queue names.

   c) Create a route node and call it TEST.

d) Connect the `IN` node to the input terminal of the TEST node.
e) Create bob, and `cecil` output terminals for the TEST node.
f) Connect the bob output terminal to the BOB node.
g) Connect the `cecil` output terminal to the `CECIL` node.
h) Connect the DEF node to the default output terminal.
i) Apply the following rules:

```
$Root/MQRFH2/usr/routeTo/text()="bob"
```

```
$Root/MQRFH2/usr/routeTo/text()="cecil"
```

9. Deploy the message flow to the IBM Integration Bus runtime component.
10. Running as the user `Alice` put a message that also contains a message property called `routeTo` with a value of either bob or `cecil`. Running the sample application **`amqsstm`** will allow you to do this.

```
Sample AMQSSTMA start
target queue is TEST.Q
Enter property name
routeTo
Enter property value
bob
Enter property name

Enter message text
My Message to Bob
Sample AMQSSTMA end
```

11. Running as user *bob* retrieve the message from the queue QBOB using the sample application **`amqsget`**.

Results

When *alice* puts a message on the `QIN` queue, the message is protected. It is retrieved in protected form by the IBM Integration Bus from the `AIN` alias queue. IBM Integration Bus decides where to route the message reading the `routeTo` property which is, as all properties, not encrypted. IBM Integration Bus places the message on the appropriate unprotected alias avoiding its further protection. When received by *bob* or *cecil* from the queue, the message is decrypted and the digital signature is verified.

Scenario 2 - Integration Bus can see message content

About this task

In this scenario, a group of individuals are allowed to send messages to IBM Integration Bus. Another group are authorized to receive messages which are created by IBM Integration Bus. The transmission between the parties and IBM Integration Bus cannot be eavesdropped.

Remember that IBM Integration Bus reads protection policies and certificates only when a queue is opened, so you must reload the execution group after making any updates to protection policies for the changes to take effect.

```
mqsireload execution-group-name
```

If IBM Integration Bus is considered an authorized party allowed to read or sign the message payload, you must configure Advanced Message Security for the user starting the IBM Integration Bus service. Be aware it is not necessarily the same user who puts/gets the messages onto queues nor the user creating and deploying the IBM Integration Bus applications.

Procedure

1. Configure *alice*, *bob*, *cecil* and *dave* and the IBM Integration Bus service user, to use Advanced Message Security as described in the **Quick Start Guide** (Windows or AIX).

   Ensure the following steps are completed:

   - Creating and authorizing users
   - Creating Key Database and Certificates
   - Creating keystore.conf
2. Provide *alice*, *bob*, *cecil* and *dave's* certificates to the IBM Integration Bus service user.

   Do this by extracting each of the certificates identifying *alice*, *bob*, *cecil* and *dave* to external files, then adding the extracted certificates to the IBM Integration Bus keystore. It is important that you use the method described in **Task 5. Sharing Certificates** in the **Quick Start Guide** (Windows or AIX).
3. Provide the IBM Integration Bus service user's certificate to *alice*, *bob*, *cecil* and *dave*.

   Do this using the method specified in the previous step.

   Note: *Alice* and *bob* need the IBM Integration Bus service user's certificate to encrypt the messages correctly. The IBM Integration Bus service user needs *alice's* and *bob's* certificates to verify authors of the messages. The IBM Integration Bus service user needs *cecil's* and *dave's* certificates to encrypt the messages for them. *cecil* and *dave* need the IBM Integration Bus service user's certificate to verify if the message comes from IBM Integration Bus.
4. Define a local queue named IN and define the security policy with *alice* and *bob* specified as authors, and the service user for the IBM Integration Bus specified as recipient:

   ```
   setmqspl -m QMgrName -p IN -s MD5 -a "CN=alice,O=IBM,C=GB" -a "CN=bob,O=IBM,C=GB"
   -e AES256 -r "CN=broker,O=IBM,C=GB"
   ```
5. Define a local queue named OUT, and define the security policy with the service user for the IBM Integration Bus specified as author, and *cecil* and *dave* specified as recipients:

   ```
   setmqspl -m QMgrName -p OUT -s MD5 -a "CN=broker,O=IBM,C=GB" -e AES256
   -r "CN=cecil,O=IBM,C=GB" -r "CN=dave,O=IBM,C=GB"
   ```
6. In IBM Integration Bus create a message flow with an `MQInput` and `MQOutput` node. Configure the `MQInput` node to use the `IN` queue and the `MQOutput` node to use the `OUT` queue.
7. Deploy the message flow to the IBM Integration Bus runtime component.
8. Running as user *alice* or *bob* put a message on the queue `IN` using the sample application **`amqsput`**.
9. Running as user *cecil* or *dave* retrieve the message from the queue `OUT` using the sample application **`amqsget`**.

Results

Messages sent by *alice* or *bob* to the input queue `IN` are encrypted allowing only IBM Integration Bus to read it. IBM Integration Bus only accepts messages from *alice* and *bob* and rejects any others. The accepted messages are appropriately processed, then signed and encrypted with *cecil's* and *dave's* keys before being put onto the output queue `OUT`. Only *cecil* and *dave* are capable of reading it, messages not signed by IBM Integration Bus are rejected.

Using Advanced Message Security with Managed File Transfer

This scenario explains how to configure Advanced Message Security to provide message privacy for data being sent through a Managed File Transfer.

Before you begin

Ensure that you have Advanced Message Security component installed on the IBM MQ installation hosting the queues used by Managed File Transfer that you want to protect.

If your Managed File Transfer agents are connecting in bindings mode, ensure you also have the IBM Global Security Kit (GSKit) component installed on their local installation.

About this task

When transfer of data between two Managed File Transfer agents is interrupted, possibly confidential data might remain unprotected on the underlying IBM MQ queues used to manage the transfer. This scenario explains how to configure and use Advanced Message Security to protect such data on the Managed File Transfer queues.

In this scenario we consider a simple topology comprising one machine with two Managed File Transfer queues and two agents, AGENT1 and AGENT2, sharing a single queue manager, as described in the scenario Managed File Transfer scenario. Both agents connect in the same way, either in bindings mode or client mode.

1. Creating certificates

Before you begin

This scenario uses a simple model where a user `ftagent` in a group FTAGENTS is used to run the Managed File Transfer Agent processes. If you are using your own user and group names, change the commands accordingly.

About this task

Advanced Message Security uses public key cryptography to sign and/or encrypt messages on protected queues.

Note:

- If your Managed File Transfer agents are running in bindings mode, the commands that you use to create a CMS (Cryptographic Message Syntax) keystore are detailed in the **Quick Start Guide** (Windows or AIX) for your platform.
- If your Managed File Transfer agents are running in client mode, the commands you will need to create a JKS (Java Keystore) are detailed in the "Quick Start Guide for AMS with Java clients" on page 614.

Procedure

1. Create a self-signed certificate to identify the user `ftagent` as detailed in the appropriate Quick Start Guide.
   Use a Distinguished Name (DN) as follows:

   ```
   CN=ftagent, OU=MFT, O=<organisation>, L=<location>, ST=<state>, C=<country>
   ```

2. Create a `keystore.conf` file to identify the location of the keystore and the certificate within it as detailed in the appropriate Quick Start Guide.

2. Configuring message protection

About this task

You should define a security policy for the data queue used by AGENT2, using the **setmqspl** command. In this scenario the same user is used to start both agents, and therefore the signer and receiver DN are the same and match the certificate we generated.

Procedure

1. Shut down the Managed File Transfer agents in preparation for protection using the **fteStopAgent** command.
2. Create a security policy to protect the SYSTEM.FTE.DATA.AGENT2 queue.

```
setmqspl -m hubQM -p SYSTEM.FTE.DATA.AGENT2 -s SHA1 -a "CN=ftagent, OU=MFT,
O=<organisation>, L=<location>, ST=<state>, C=<country>"
-e AES128 -r "CN=ftagent, OU=MFT, O=<organisation>, L=<location>, ST=<state>, C=<country>"
```

3. Ensure the user running the Managed File Transfer Agent process has access to browse the system policy queue and put messages on the error queue.

```
setmqaut -m hubQM -t queue -n SYSTEM.PROTECTION.POLICY.QUEUE -p ftagent +browse
```

```
setmqaut -m hubQM -t queue -n SYSTEM.PROTECTION.ERROR.QUEUE -p ftagent +put
```

4. Restart your Managed File Transfer agents using the **`fteStartAgent`** command.
5. Confirm that your agents restarted successfully by using the **`fteListAgents`** command and verifying that the agents are in READY status.

Results

You are now able to submit transfers from AGENT1 to AGENT2, and the file contents will be transmitted securely between the two agents.

Advanced Message Security installation overview

Install the Advanced Message Security component on various platforms.

Procedure

- Multi
  Install Advanced Message Security on Multiplatforms.
- z/OS
  Install IBM MQ Advanced for z/OS.
- z/OS
  Install IBM MQ Advanced for z/OS Value Unit Edition.

Related tasks

Uninstalling Advanced Message Security

z/OS Auditing for AMS on z/OS

Advanced Message Security (AMS) for z/OS provides a means for optional auditing of operations by applications on policy protected queues. When enabled, IBM System Management Facility (SMF) audit records are generated for the success and failure of these operations on policy-protected queues. Operations audited include MQPUT, MQPUT1, and MQGET.

Auditing is disabled by default, however, you can activate auditing by configuring _AMS_SMF_TYPE and _AMS_SMF_AUDIT in the configured Language Environment® _CEE_ENVFILE file for the AMS address space. For more information, see Create procedures for Advanced Message Security. The _AMS_SMF_TYPE variable is used to designate the SMF record type and is a number between 128 and 255. A SMF record type of 180 is usual, however is not mandatory. Auditing is disabled by specifying a value of 0. The _AMS_SMF_AUDIT variable configures whether audit records are created for operations that are successful, operations that fail, or both. The auditing options can also be dynamically changed while AMS is active using operator commands. For more information, see Operating Advanced Message Security.

The SMF record is defined using subtypes, with subtype 1 being a general auditing event. The SMF record contains all data relevant to the request being processed.

The SMF record is mapped by the CSQ0KSMF macro (note the zero in the macro name), which is provided in the target library SCSQMACS. If you are writing data-reduction programs for SMF data, you can include this mapping macro to aid in the development and customization of SMF post-processing routines.

In the SMF records produced by Advanced Message Security for z/OS, the data is organized into sections. The record consists of:

- a standard SMF header
- a header extension defined by Advanced Message Security for z/OS
- a product section
- a data section

The product section of the SMF record is always present in the records produced by Advanced Message Security for z/OS. The data section varies based on subtype. Currently, one subtype is defined and therefore a single data section is used.

SMF is described in the z/OS System Management Facilities manual (SA22-7630). Valid record types are described in the SMFPRMxx member of your system PARMLIB data set. See SMF documentation for more information.

Advanced Message Security audit report generator (CSQ0USMF)

Advanced Message Security for z/OS provides an audit report generator tool called CSQ0USMF which is provided in the installation SCSQAUTH library. Sample JCL to run the CSQ0USMF utility called CSQ40RSM is provided in the installation library SCSQPROC.

Before running the CSQ0USMF utility, the SMF type 180 records must be dumped from the system SMF data sets to a sequential data set. As an example, this JCL dumps SMF type 180 records from an SMF data set, and transfers them to a target data set:

```
//IFAUDUMP EXEC PGM=IFASMFDP
//INDD1 DD DSN=SYSn.MANn.syst,DISP=SHR
//OUTDD1 DD DSN=your.target.dataset,DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
INDD(INDD1,OPTIONS(DUMP))
OUTDD(OUTDD1,TYPE(180))
/*
```

You must verify the actual SMF data set names used by your installation. The target data set for the dumped records must have a record format of VBS, and a record length of 32760.

Note: If SMF logstreams are being used, you must use program IFASMFDL to dump a logstream out to a sequential dataset. See Processing type 116 SMF records for an example of the JCL used.

The target data set can then be used as input to the CSQ0USMF utility to produce an AMS audit report. For example:

```
//STEP1 EXEC PGM=CSQ0USMF,
// PARM=('/ -SMFTYPE 180 -M qmgr')
//STEPLIB DD DSN=thlqual.SCSQANLE,DISP=SHR
//    DD DSN=thlqual.SCSQAUTH,DISP=SHR
//SMFIN DD DSN=your.target.dataset,DISP=SHR
//
```

The CSQ0USMF program accepts two optional parameters, which are listed in Table 103 on page 627:

Table 103. CSQ0USMF optional parameters

Parameter	Value	Description
SMFTYPE	nnn	The SMF record type applicable to the audit report. The CSQ0USMF program uses only SMF records that match the SMFTYPE value when generating the report. If you do not specify SMFTYPE, a default value of 180 is used.
M	qmgr	The IBM MQ queue manager name applicable to the audit report. If you do not specify the -M parameter, the audit report will include all audit records for all queue managers represented in the SMFIN data set.

Using keystores and certificates with AMS

To provide transparent cryptographic protection to IBM MQ applications, Advanced Message Security uses the keystore file, where public key certificates and a private key are stored. On z/OS, a SAF key ring is used instead of a keystore file.

In Advanced Message Security, users and applications are represented by public key infrastructure (PKI) identities. This type of identity is used to sign and encrypt messages. The PKI identity is represented by the subject's **distinguished name (DN)** field in a certificate that is associated with signed and encrypted messages. For a user or application to encrypt their messages they require access to the keystore file where certificates and associated private and public keys are stored.

ALW On AIX, Linux, and Windows, the location of the keystore is provided in the keystore configuration file, which is `keystore.conf` by default. Each Advanced Message Security user must have the keystore configuration file that points to a keystore file. Advanced Message Security accepts the following format of keystore files: `.kdb`, `.jceks`, `.jks`.

The default location of the `keystore.conf` file is:

- IBM i Linux AIX 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 specify this with the **MQS_KEYSTORE_CONF** environment variable, as shown in the following example commands:

- For Java: `java -D`*`MQS_KEYSTORE_CONF=path/filename app_name`*
- For a C client and server:
  - Linux AIX On AIX and Linux: `export MQS_KEYSTORE_CONF=`*`path/filename`*
  - Windows On Windows: `set MQS_KEYSTORE_CONF=`*`path\filename`*

    Note: The path on Windows can, and should, specify the drive letter if more than one drive letter is available.

Protecting sensitive information in the `keystore.conf` file

In order to access keystore file sensitive information, such as passwords, you must supply tokens so that IBM MQ Advanced Message Security (AMS) can access the keystore and sign and encrypt messages.

You should protect the sensitive information contained in the keystore configuration file using the **runamscred** command provided with AMS. See "Setting up AMS password protection for configuration files" on page 645 for details on how to protect configuration files.

When protecting passwords, you should use a custom, strong encryption key. In order to access the passwords during runtime, this encryption key must be supplied to AMS.

There are two methods of supplying the location of the encryption key file, which are, through the:

- **amscred.keyfile** configuration property in the `keystore.conf` file
- **MQS_AMSCRED_KEYFILE** environment variable

The order of precedence is **MQS_AMSCRED_KEYFILE**, followed by **amscred.keyfile**, and then the default key.

Related concepts

"Sender distinguished names in AMS" on page 655
The sender distinguished names (DNs) identify users who are authorized to place messages on a queue. A sender uses their certificate to sign a message, prior to placing the message on a queue.

"Recipient distinguished names in AMS" on page 656
The recipient distinguished names (DN) identify users who are authorized to retrieve messages from a queue.

Structure of the keystore configuration file (keystore.conf) for AMS

The keystore configuration file (`keystore.conf`) points Advanced Message Security to the location of the appropriate keystore.

Each of the following configuration file types has a prefix:

AMSCRED
: Parameters that relate to the password protection system.

CMS
: Certificate Management System, configuration entries are prefixed with: `cms.`

PKCS#11
: Public Key Cryptography Standard #11, configuration entries are prefixed with: `pkcs11.`

IBM i **PEM**
: Privacy Enhanced Mail format, configuration entries are prefixed with: `pem.`

JKS
: Java KeyStore, configuration entries are prefixed with: `jks.`

JCEKS
: Java Cryptographic Encryption KeyStore, configuration entries are prefixed with: `jceks.`

z/OS MQ Adv. VUE **JCERACFKS**
: Java Cryptographic Encryption RACF keyring KeyStore, configuration entries are prefixed with: `jceracfks`.

Important: From IBM MQ 9.0 the `JCEKS.provider` and `JKS.provider` values are ignored. The Bouncy Castle provider is used, in conjunction with whichever JCE/JCE provision is supplied by the JRE in use. For more information, see "Support for non-IBM JREs with AMS" on page 632.

Example structures for keystores:

CMS

```
cms.keystore = /dir/keystore_file
cms.certificate = certificate_label
```

PKCS#11

```
pkcs11.library = dir\cryptoki.dll
pkcs11.certificate = certificatelabel
pkcs11.token = tokenlabel
pkcs11.token_pin = tokenpin
pkcs11.secondary_keystore = dir\signers
pkcs11.encrypted = no
```

IBM i PEM

```
pem.private = /dir/keystore_file_private_key
pem.public = /dir/keystore_file_public_keys
pem.password = password
pem.encrypted = no
```

Java JKS

```
jks.keystore = dir/Keystore
jks.certificate = certificate_label
jks.encrypted = no
jks.keystore_pass = password
jks.key_pass = password
```

Java JCEKS

```
jceks.keystore = dir/Keystore
jceks.certificate = certificate_label
jceks.encrypted = no
jceks.keystore_pass = password
jceks.key_pass = password
```

Java JCERACFKS

```
jceracfks.keystore = safkeyring://user/keyring
jceracfks.certificate = certificate_label
```

Java PKCS#11

```
pkcs11.library = dir\cryptoki.dll
pkcs11.certificate = certificatelabel
pkcs11.token = tokenlabel
pkcs11.token_pin = tokenpin
pkcs11.secondary_keystore = dir\signers
pkcs11.secondary_keystore_pass = password
pkcs11.encrypted = no
```

Table 104. Summary of parameters needed for each configuration file type

		Configuration file type				
Parameters	**Required**	**Java (PKCS#11, JKS, JCEKS, and JCERACFKS)**	IBM i **PEM**	**PKCS#11**	**CMS**	**AMSCRED**
`keystore`	✓	✓			✓	
IBM i `private`	✓		IBM i ✓			

Table 104. Summary of parameters needed for each configuration file type (continued)

Parameters	Required	Configuration file type				
		Java (PKCS#11, JKS, JCEKS, and JCERACFKS)	IBM i PEM	PKCS#11	CMS	AMSCRED
IBM i `public`	✓		IBM i ✓			
IBM i `password`	✓		IBM i ✓			
`library`	✓	✓		✓		
`certificate`	✓	✓		✓	✓	
`token`	✓	✓		✓		
`token_pin`	✓	✓		✓		
`secondary_keystore`	✓	✓		✓		
`secondary_keystore_password`	✓	✓				
`encrypted`		✓	IBM i ✓	✓		
`keystore_pass`	✓	✓				
`key_pass`		✓				
`provider`		✓				
`keyfile`						✓You

Note that you can add comments using the # symbol.

Configuration file parameters are defined as follows:

keystore

CMS and Java configuration only.

Path to the keystore file for CMS, JKS, and JCEKS configuration.

z/OS MQ Adv. VUE URI to the RACF keyring for JCERACFKS configuration.

Important:

- The path to the keystore file must not include the file extension.
- z/OS MQ Adv. VUE The URI to the RACF keyring must be in the form:

```
safkeyring://user/keyring
```

where:

- *user* is the user id that owns the keyring
- *keyring* is the keyring name.

IBM i **private**
: PEM configuration only.

  File name of a file that contains private key and certificate in PEM format.

IBM i **public**
: PEM configuration only.

  File name of a file that contains trusted public certificates in PEM format.

IBM i **password**
: PEM configuration only.

  Password that is used to decrypt an encrypted private key.

  You should protect this field using the native AMS password protection tool; see "Protecting passwords" on page 632

library
: PKCS#11 only.

  Path name of the PKCS#11 library.

certificate
: CMS, PKCS#11 and Java configuration only.

  Certificate label.

token
: PKCS#11 only.

  Token label.

token_pin
: PKCS#11 only.

  PIN to unlock the token.

  For Java operations only; you should protect this field using the Java AMS password protection tool; see "Protecting passwords" on page 632.

  For Native operations only; you should protect this field using the native AMS password protection tool; see "Protecting passwords" on page 632.

secondary_keystore
: PKCS#11 only.

  Path name of the CMS keystore, provided without the `.kdb` extension, that contains anchor certificates (root certificates) required by certificates stored on the PKCS #11 token. The secondary keystore can also contain certificates that are intermediate in the trust chain, as well as recipient certificates that are defined in the privacy security policy. This CMS keystore must be accompanied by a stash file which must be located in the same directory as the secondary keystore.

  For Java environments a JKS keystore is required and you must provide a **secondary_keystore_password**.

secondary_keystore_password
: Java PKCS#11 only.

  Password for the JKS keystore provided through the `secondary_keystore` property. You should protect this field using the Java AMS password protection tool; see "Protecting passwords" on page 632.

encrypted
: Java and, from IBM MQ 9.3.0, PKCS#11 and IBM i PEM only.

  Status of the password.

keystore_pass
Java configuration only.

Password for the keystore file.

For Java operations only. You should protect this field using the Java AMS password protection tool; see "Protecting passwords" on page 632.

key_pass
Java configuration only.

Password for the private key of the user.

For Java operations only; you should protect this field using the Java AMS password protection tool; see "Protecting passwords" on page 632.

keyfile
Provides the location of the initial key to use when protecting or decrypting passwords contained in this configuration file; see "Protecting passwords" on page 632

provider
Java configuration only.

The Java security provider that implements cryptographic algorithms required by the keystore certificate.

Important: Information that is stored in the keystore is crucial for the secure flow of data that is sent by using IBM MQ. Security administrators must pay particular attention when they are assigning file permissions to these files.

Protecting passwords

You should protect the passwords and other sensitive information contained in the `keystore.conf` file. For more information, see **runamscred**.

Example of the `keystore.conf` file:

```
# Native AMS application configuration
cms.keystore = c:\Documents and Settings\Alice\AliceKeystore
cms.certificate = AliceCert

# Java AMS application configuration
jceks.keystore = c:/Documents and Settings/Alice/AliceKeystore
jceks.certificate = AliceCert
jceks.encrypted = no
jceks.keystore_pass = passw0rd
jceks.key_pass = passw0rd
jceks.provider = IBMJCE
```

Related tasks
"Setting up AMS password protection for configuration files" on page 645
Storing keystore and private key passwords as plain text poses a security risk so Advanced Message Security provides a tool that can scramble those passwords using a user's key.

Support for non-IBM JREs with AMS

IBM MQ classes for Java and IBM MQ classes for JMS support Advanced Message Security operation when running with non-IBM JREs.

Advanced Message Security (AMS) implements Cryptographic Message Syntax (CMS). The CMS syntax is used to digitally sign, digest, authenticate, or encrypt arbitrary message content.

From IBM MQ 9.0, the Advanced Message Security support in IBM MQ classes for Java and IBM MQ classes for JMS uses the open source Bouncy Castle packages to support CMS. This means that these classes can support Advanced Message Security operation when running with non-IBM JREs.

Before IBM MQ 9.0, Advanced Message Security was not supported in non-IBM JREs in Java clients. Advanced Message Security support in the IBM MQ classes for Java and IBM MQ classes for JMS depended on CMS support specifically provided by the IBM implementation of the Java Cryptography Extensions (JCE). Because of this restriction, the functionality was only available when using a Java runtime environment (JRE) that included the Java JCE provider.

Location and version numbering for Bouncy Castle JAR files

The Bouncy Castle JAR files that are needed for support for non-IBM JREs are included as part of the IBM MQ classes for Java and IBM MQ classes for JMS installation package.

The Bouncy Castle JAR files used are the following files:

The provider JAR file, which is fundamental to Bouncy Castle operations.
V 9.4.0 From IBM MQ 9.4.0, this JAR file is called `bcprov-jdk18on.jar`.

The "PKIX" JAR file, which contains the support for CMS operations that are used by Advanced Message Security.
V 9.4.0 From IBM MQ 9.4.0, this JAR file is called `bcpkix-jdk18on.jar`.

The "util" JAR file, which contains classes used by the other Bouncy Castle JAR files.
V 9.4.0 From IBM MQ 9.4.0, this JAR file is called `bcutil-jdk18on.jar`.

Dependencies

The IBM MQ 9.1 and later classes have been tested with IBM JREs and Oracle JREs. They are also likely to run successfully under any J2SE-compliant JRE. However, you should note the following dependencies:

- There are no changes to Advanced Message Security configuration.
- The Bouncy Castle classes are used only for CMS operations. All other security-related operations, for example keystore access, the actual encryption of data, and calculation of signature checksums use the functionality that is provided by the JRE.

  Important: For this reason, the JRE used must include a JCE provider implementation.
- To use some *strong* encryption algorithms, you might need to install the *unrestricted* policy files for the JRE's JCE implementation.

  Refer to the JRE documentation for more details.
- If you have enabled Java security:
  - Add `java.security.SecurityPermissioninsertProvider.BC` to the application so that the Bouncy Castle classes can be used as a security provider.
  - Grant `java.security.AllPermission` to the Bouncy Castle JAR files.

  V 9.4.0 From IBM MQ 9.4.0, these files are:

  mq_install_dir/java/lib/bcutil-jdk18on.jar
  mq_install_dir/java/lib/bcpkix-jdk18on.jar
  mq_install_dir/java/lib/bcprov-jdk18on.jar

Related concepts
What is installed for IBM MQ classes for JMS
What is installed for IBM MQ classes for Java

Multi Message Channel Agent (MCA) interception and AMS

MCA interception enables a queue manager running under IBM MQ to selectively enable policies to be applied for server connection channels.

MCA interception allows clients that remain outside AMS to still be connected to a queue manager and their messages to be encrypted and decrypted.

MCA interception is intended to provide AMS capability when AMS cannot be enabled at the client. Note that using MCA interception and an AMS-enabled client leads to double-protection of messages which might be problematic for receiving applications. For more information, see "Disabling Advanced Message Security at the client" on page 636.

Note: MCA interceptors are not supported for AMQP or MQTT channels.

Keystore configuration file

By default, the keystore configuration file for MCA interception is `keystore.conf` and is located in the `.mqs` directory in the HOME directory path of the user who started the queue manager or the listener. The keystore can also be configured by using the MQS_KEYSTORE_CONF environment variable. For more information about configuring the AMS keystore, see "Using keystores and certificates with AMS" on page 627.

To enable MCA interception, you must provide the name of a channel that you want to use in the keystore configuration file. For MCA Interception, only a cms keystore type can be used.

See "MCA interception example for AMS" on page 634 for an example of setting up MCA interception.

Attention: You must complete client authentication and encryption on the selected channels, for example, by using SSL and SSLPEER or CHLAUTH TYPE(SSLPEERMAP), to ensure that only authorized clients can connect and use this capability.

IBM i

If your enterprise uses IBM i, and you selected a commercial Certificate Authority (CA) to sign your certificate, the Digital Certificate Manager creates a certificate request in PEM (Privacy-Enhanced Mail) format. You must forward the request to your chosen CA.

To do this, you must use the following command to select the correct certificate for the channel specified in `channelname`:

```
pem.certificate.channel.channelname
```

MCA interception example for AMS

An example task on how you set up an AMS MCA interception.

Before you begin

Attention: You must complete client authentication and encryption on the selected channels, for example, by using SSL and SSLPEER or CHLAUTH TYPE(SSLPEERMAP), to ensure that only authorized clients can connect and use this capability.

If your enterprise uses IBM i, and you selected a commercial Certificate Authority (CA) to sign your certificate, the Digital Certificate Manager creates a certificate request in PEM (Privacy-Enhanced Mail) format. You must forward the request to your chosen CA.

About this task

This task takes you through the process of setting up your system to use MCA interception, then verifying the setup.

Note: IBM MQ, includes the AMS interceptors and dynamically enables them in the MQ client and server runtime environments.

Attention:

- Replace `userID` in the code with your user ID.
- The following procedure does not work as expected in IBM MQ unless the AMS interception is deactivated on the client.

Procedure

1. Create the key database and certificates by using the following commands to create a shell script.

   Also, change the **`INSTLOC`** and **`KEYSTORELOC`** or run the required commands. Note that you might not need to create the certificate for bob.

```
INSTLOC=/opt/mqm
KEYSTORELOC=/home/userID/var/mqm
mkdir -p $KEYSTORELOC
chmod -R 777 $KEYSTORELOC
chown -R mqm:mqm $KEYSTORELOC
export PATH=$PATH:$INSTLOC/gskit8/bin
echo "PATH = $PATH"
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$INSTLOC/gskit8/lib64

runmqakm -keydb -create -db $KEYSTORELOC/alicekey.kdb -pw passw0rd -stash
runmqakm -keydb -create -db $KEYSTORELOC/bobkey.kdb -pw passw0rd -stash
runmqakm -cert -create -db $KEYSTORELOC/alicekey.kdb -pw passw0rd \
-label alice_cert -dn "cn=alice,O=IBM,c=IN" -default_cert yes
runmqakm -cert -create -db $KEYSTORELOC/bobkey.kdb -pw passw0rd \
-label bob_cert  -dn "cn=bob,O=IBM,c=IN"  -default_cert yes
```

2. Share the certificates between the two key databases so that each user can successfully identify the other.

   It is important that you use the method described for sharing certificates in the *Quick Start Guide*, for the platform your enterprise uses:

   Windows
   Task 5 Sharing certificates

   AIX and Linux
   Task 5 Sharing certificates

   Java clients
   Task 5 Sharing certificates

3. Create `keystore.conf` with the following configuration: `Keystore.conf location: /home/userID/ssl/ams1/`

```
cms.keystore = /home/userID/ssl/ams1/alicekey
cms.certificate.channel.SYSTEM.DEF.SVRCONN = alice_cert
```

   Attention:

   a. The keystore must be on the system where the queue manager is.
   b. You must specify a specific channel for `cms.certificate` to enable MCA intervention, and then the queue manager performs AMS operations on applications connecting through that channel to queues with policies set.

4. Create and start queue manager AMSQMGR1
5. Define a TCP listener using an available port number under QMGR control.

   For example:

   ```
   DEFINE LISTENER(MY.LISTENER) TRPTYPE(TCP) PORT(14567) CONTROL(QMGR)
   ```

6. Start the listener and verify that it started correctly.

   For example:

   ```
   START  LISTENER(MY.LISTENER)
   DISPLAY LSSTATUS(MY.LISTENER) PORT
   ```

7. Stop the queue manager.
8. Set the keystore with the **`MQS_KEYSTORE_CONF`** environment variable:

```
export MQS_KEYSTORE_CONF=/home/userID/ssl/ams1/keystore.conf
```

9. Start the queue manager on the same shell, so that the **MQS_KEYSTORE_CONF** environment variable is available to the queue manager.
10. Set the security policy and verify:

```
setmqspl -m AMSQMGR1 -s SHA256 -e AES256 -p TESTQ -a "CN=alice,O=IBM,C=IN" \
-r "CN=alice,O=IBM,C=IN"
dspmqspl -m AMSQMGR1
```

    See setmqspl and dspmqspl for more information.
11. Set the **MQSERVER** environment variable:

```
export MQSERVER='SYSTEM.DEF.SVRCONN/TCP/127.0.0.1(14567)'
```

12. Remove the security policy and verify the result:

```
setmqspl -m AMSQMGR1 -p TESTQ -remove
dspmqspl -m AMSQMGR1
```

13. Browse the queue from your IBM MQ 9.4 installation:

```
/opt/mq94/samp/bin/amqsbcg TESTQ AMSQMGR1
```

    The browse output shows the messages in encrypted format.
14. Set the security policy and verify the result:

```
setmqspl -m AMSQMGR1 -s SHA256 -e AES256 -p TESTQ -a "CN=alice,O=IBM,C=IN"
-r "CN=alice,O=IBM,C=IN"
dspmqspl -m AMSQMGR1
```

15. Run **amqsgetc** from your IBM MQ 9.4 installation:

```
/opt/mqm/samp/bin/amqsgetc TESTQ TESTQMGR
```

Related concepts

"Structure of the keystore configuration file (keystore.conf) for AMS" on page 628
The keystore configuration file (`keystore.conf`) points Advanced Message Security to the location of the appropriate keystore.

Related reference

"Known limitations of AMS" on page 587
There are a number of IBM MQ options that are either not supported, or have limitations for Advanced Message Security.

Disabling Advanced Message Security at the client

You need to disable IBM MQ Advanced Message Security (AMS) if you are using an IBM MQ client to connect to a queue manager from an earlier version of the product and a 2085 (MQRC_UNKNOWN_OBJECT_NAME) error is reported.

About this task

IBM MQ Advanced Message Security (AMS) is automatically enabled in an IBM MQ client and so, by default, the client tries to check the security policies for objects at the queue manager.

If this error is reported, when you are trying to connect to a queue manager from an earlier version of the product, you can disable AMS as follows:

- For Java clients, in any of the following ways:
  - By setting an environment variable **AMQ_DISABLE_CLIENT_AMS**.
  - By setting the Java system property com.ibm.mq.cfg.AMQ_DISABLE_CLIENT_AMS.
  - By using the **DisableClientAMS** property, under the Security stanza in the `mqclient.ini` file.
- For C clients, by setting an environment variable **MQS_DISABLE_ALL_INTERCEPT**.

Note: You cannot use the **`AMQ_DISABLE_CLIENT_AMS`** environment variable for C clients. You need to use the **`MQS_DISABLE_ALL_INTERCEPT`** environment variable instead.

Procedure

- To disable AMS at the client, use one of the following options:

  AMQ_DISABLE_CLIENT_AMS environment variable

  You need to set this variable in the following cases:

  - If you are using a Java runtime environment (JRE) other than the IBM Java runtime environment (JRE)
  - If you are using an IBM MQ IBM MQ classes for JMS or IBM MQ classes for Java client.

  Create the **`AMQ_DISABLE_CLIENT_AMS`** environment variable and set it to TRUE in the environment where the application is running. For example:

  ```
  export AMQ_DISABLE_CLIENT_AMS=TRUE
  ```

  Java system property com.ibm.mq.cfg.AMQ_DISABLE_CLIENT_AMS

  For IBM MQ classes for JMS and IBM MQ classes for Java clients, you can set the Java system property com.ibm.mq.cfg.AMQ_DISABLE_CLIENT_AMS to the value TRUE for the Java application.

  For example, you can set the Java system property as a `-D` option when the Java command is invoked:

  JMS 3.0
  ```
  java -Dcom.ibm.mq.cfg.AMQ_DISABLE_CLIENT_AMS=TRUE -cp <MQ_INSTALLATION_PATH>/
  java/lib/com.ibm.mq.jakarta.client.jar  my.java.applicationClass
  ```

  JMS 2.0
  ```
  java -Dcom.ibm.mq.cfg.AMQ_DISABLE_CLIENT_AMS=TRUE -cp <MQ_INSTALLATION_PATH>/
  java/lib/com.ibm.mq.allclient.jar  my.java.applicationClass
  ```

  Alternatively, you can specify the Java system property within a JMS configuration file, `jms.config`, if the application uses this file.

  MQS_DISABLE_ALL_INTERCEPT environment variable

  You need to set this environment variable if you are using IBM MQ with native clients and you need to disable AMS at the client.

  Create the environment variable **`MQS_DISABLE_ALL_INTERCEPT`** and set it to TRUE in the environment where the client is running. For example:

  ```
  export MQS_DISABLE_ALL_INTERCEPT =TRUE
  ```

  You can use the **`MQS_DISABLE_ALL_INTERCEPT`** environment variable for C clients only. For Java clients, you need to use the **`AMQ_DISABLE_CLIENT_AMS`** environment variable instead.

  `DisableClientAMS` property in the `mqclient.ini` file

  You can use this option for IBM MQ classes for JMS and IBM MQ classes for Java clients, and for C clients.

  Add the property name `DisableClientAMS` under the **`Security`** stanza the `mqclient.ini` file as shown in the following example:

  ```
  Security:
  DisableClientAMS=Yes
  ```

  You can also enable AMS as shown in the following example:

  ```
  Security:
  DisableClientAMS=No
  ```

What to do next

For more information on problems with opening AMS protected queues, see Problems opening protected queues when using AMS with JMS.

Related concepts
"Message Channel Agent (MCA) interception and AMS" on page 633
MCA interception enables a queue manager running under IBM MQ to selectively enable policies to be applied for server connection channels.

Related tasks
IBM MQ MQI client configuration file, `mqclient.ini`

Related reference
The IBM MQ classes for JMS configuration file

Certificate requirements for AMS

Certificates must have an RSA public key in order to be used with Advanced Message Security.

For more information about different public key types and how to create them, see "Digital certificates and CipherSpec compatibility in IBM MQ" on page 47.

Key usage extensions

Key usage extensions place additional restrictions on the way a certificate can be used.

In Advanced Message Security, the key usage of X.509 v3 certificates must be set in accordance with the RFC 5280 specification.

For the quality of protection integrity, if certificate key usage extensions are set, that set must include at least one of the two:

- **`nonRepudiation`**
- **`digitalSignature`**

For the quality of protection privacy, if certificate key usage extensions are set, that set must include:

- **`keyEncipherment`**

For the quality of protection confidentiality, if certificate key usage extensions are set, that set must include:

- **`dataEncipherment`**

Extended key usage further refines key usage extensions. For all qualities of protection, if certificate extended key usage is set, the set must include:

- **`emailProtection`**

Related concepts
"Quality of protection in AMS" on page 658
Advanced Message Security data-protection policies imply a quality of protection (QOP).

Certificate validation methods in AMS

You can use Advanced Message Security to detect and reject revoked certificates so that messages on your queues are not protected using certificates that do not fulfill security standards.

AMS allows you to verify a certificate validity by using either Online Certificate Status Protocol (OCSP) or certificate revocation list (CRL).

AMS can be configured for either OCSP or CRL checking or both. If both methods are enabled, then, for performance reasons, AMS uses OCSP for revocation status first. If the revocation status of a certificate is undetermined after the OCSP checking, AMS uses the CRL checking.

Note that both OCSP and CRL checking are enabled by default.

Related concepts

"Online Certificate Status Protocol (OCSP) in AMS" on page 639
Online Certificate Status Protocol (OCSP) determines whether a certificate has been revoked and, therefore, helps to determine whether the certificate can be trusted. OCSP is enabled by default.

"Certificate revocation lists (CRLs) in AMS" on page 641
CRLs holds a list of certificates that have been marked by Certificate Authority (CA) as no longer trusted for a variety of reasons, for example, the private key has been lost or compromised.

Online Certificate Status Protocol (OCSP) in AMS

Online Certificate Status Protocol (OCSP) determines whether a certificate has been revoked and, therefore, helps to determine whether the certificate can be trusted. OCSP is enabled by default.

OCSP is not supported on IBM i sytems.

Enabling OCSP checking in native interceptors of Advanced Message Security
Online Certificate Status Protocol (OCSP) checking in Advanced Message Security is enabled by default, based on information in the certificates being used.

Procedure

Add the following options to the keystore configuration file:

Note: All the OCSP stanza are optional and can be specified independently.

Option	Description
`ocsp.enable=off`	Enable the OCSP checking if the certificate being checked has an Authority Info Access (AIA) Extension with an PKIX_AD_OCSP access method containing a URI of where the OCSP Responder is located. Possible values: on or `off`.
`ocsp.url=`*`responder_URL`*	The URL address of OCSP responder. If this option is omitted then non-AIA OCSP checking is disabled.
`ocsp.http.proxy.host=`*`OCSP_proxy`*	The URL address of the OCSP proxy server. If this option is omitted then a proxy is not used for non-AIA online certificate checks.
`ocsp.http.proxy.port=`*`port_number`*	The OCSP proxy server's port number. If this option is omitted then the default port of 8080 is used.
`ocsp.nonce.generation=on/off`	Generate nonce when querying OCSP. The default value is `off`.
`ocsp.nonce.check=on/off`	Check nonce after receiving a response from OCSP. The default value is `off`.
`ocsp.nonce.size=8`	Nonce size in bytes.
`ocsp.http.get=on/off`	Specify HTTP GET as your request method. If this option is set to `off`, HTTP POST is used. The default value is `off`.
`ocsp.max_response_size=20480`	Maximum size of response from the OCSP responder provided in bytes.

Option	Description
`ocsp.cache_size=100`	Enable internal OCSP response caching and set the limit for the number of cache entries.
`ocsp.timeout=30`	Waiting time for a server response, in seconds, after which Advanced Message Security times-out.
`ocsp.unknown=ACCEPT`	Defines the behavior when an OCSP server cannot be reached within a timeout period. Possible values: • `ACCEPT` Allows the certificate • `WARN` Allows the certificate and logs a warning • `REJECT` Prevents the certificate from being used and logs an error

Enabling OCSP checking in Java in AMS
To enable OCSP checking for Java in Advanced Message Security, modify the `java.security` file or the keystore configuration file.

About this task

There are two ways of enabling OCSP checking in Advanced Message Security:

Using java.security
Check whether your certificate contains an Authority Information Access (AIA) certificate extension.

Procedure

1. If AIA is not set up or you want to override your certificate, edit the `$JAVA_HOME/lib/security/java.security` file with the following properties:

   ```
   ocsp.responderURL=http://url.to.responder:port
   ocsp.responderCertSubjectName=CN=Example CA,O=IBM,C=US
   ```

   and enable OCSP checking by editing the `$JAVA_HOME/lib/security/java.security` file with the following line:

   ```
   ocsp.enable=true
   ```

2. If AIA is set up, enable OCSP checking by editing the `$JAVA_HOME/lib/security/java.security` file with the following line:

   ```
   ocsp.enable=true
   ```

What to do next

If you are using Java Security Manager, too complete the configuration, add the following Java permission to `lib/security/java.policy`

```
permission java.security.SecurityPermission "getProperty.ocsp.enable";
```

Using keystore.conf

Procedure

Add the following attribute to the configuration file:

```
ocsp.enable=true
```

Important: Setting this attribute in the configuration file overrides java.security settings.

What to do next

To complete the configuration, add the following Java permissions to `lib/security/java.policy`:

```
permission java.security.SecurityPermission "getProperty.ocsp.enable";
permission java.security.SecurityPermission "setProperty.ocsp.enable";
```

Certificate revocation lists (CRLs) in AMS

CRLs holds a list of certificates that have been marked by Certificate Authority (CA) as no longer trusted for a variety of reasons, for example, the private key has been lost or compromised.

To validate certificates, Advanced Message Security constructs a certificate chain that consists of the signer's certificate and the certificate authority's (CA's) certificate chain up to a trust anchor. A trust anchor is a trusted keystore file that contains a trusted certificate or a trusted root certificate that is used to assert the trust of a certificate. AMS verifies the certificate path using a PKIX validation algorithm. When the chain is created and verified, AMS completes the certificate validation which includes validating the issue and expiry date of each certificate in the chain against the current date, checking if the key usage extension is present in the End Entity certificate. If the extension is appended to the certificate, AMS verifies whether **digitalSignature** or **nonRepudiation** are also set. If they are not, the `MQRC_SECURITY_ERROR` is reported and logged. Next, AMS downloads CRLs from files or from LDAP depending on what values were specified in the configuration file. Only CRLs that are encoded in DER format are supported by AMS. If no CRL related configuration is found in the keystore configuration file, AMS performs no CRL validity check. For each CA certificate, AMS queries LDAP for CRLs using Distinguished Names of a CA to find its CRL. The following attributes are included in the LDAP query:

```
certificateRevocationList,
certificateRevocationList;binary,
authorityRevocationList,
authorityRevocationList;binary
deltaRevocationList
deltaRevocationList;binary,
```

Note: `deltaRevocationList` is supported only when it is specified as distribution points.

Enabling certificate validation and certificate revocation list support in native interceptors
You must modify the keystore configuration file so that Advanced Message Security can download CLRs from the Lightweight Directory Access Protocol (LDAP) server.

About this task

IBM i Enabling certificate validation and certificate revocation list support in native interceptors is not supported for Advanced Message Security on IBM i.

Procedure

Add the following options to the configuration file:

Note: All the CRL stanza are optional and can be specified independently.

Option	Description
`crl.ldap.host=`*`host_name`*	LDAP server host name.

Option	Description
`crl.ldap.port=`*`port_number`*	LDAP server port number. You can specify up to 11 servers. Multiple LDAP hosts are used to ensure transparent failover in case of LDAP connection failure. It is expected that all LDAP servers are replicas and contain the same data. When the AMS Java interceptor successfully connects to an LDAP server, it does not attempt to download CRLs from the remaining servers provided.
`crl.cdp=off`	Use this option to check or use CRLDistributionPoints extensions in certificates.
`crl.ldap.version=3`	LDAP protocol version number. Possible values: 2 or 3.
`crl.ldap.user=cn=`*`username`*	Log in to the LDAP server. If this value is not specified, CRL attributes in LDAP must be world-readable
`crl.ldap.pass=`*`password`*	Password for the LDAP server.
`crl.ldap.encrypted=`*`no/yes`*	Whether the `clr.ldap.pass` is encrypted or not. See Protecting passwords in AMS configuration files for more information.
`crl.ldap.cache_lifetime=0`	LDAP cache lifetime in seconds. Possible values: 0-86400.
`crl.ldap.cache_size=50`	LDAP cache size. This option can be specified only if the `crl.ldap.cache_lifetime` value is larger than `0`.
`crl.http.proxy.host=some.host.com`	Http proxy server port for CDP CRL retrieval.
`crl.http.proxy.port=8080`	Http proxy server port number.
`crl.http.max_response_size=204800`	The maximum size of CRL, in bytes, that can be retrieved from an HTTP server that is accepted by IBM Global Security Kit (GSKit).
`crl.http.timeout=30`	Waiting time for a server response, in seconds, after which AMS times outs.
`crl.http.cache_size=0`	HTTP cache size, in bytes.
`crl.unknown=ACCEPT`	Defines the behavior when a CRL server cannot be reached within a timeout period. Possible values: • ACCEPT Allows the certificate • WARN Allows the certificate and logs a warning • REJECT Prevents the certificate from being used and logs an error

Enabling certificate revocation list support in Java in AMS
To enable CRL support in Advanced Message Security, you must modify the keystore configuration file to allow AMS to download CRLs from the Lightweight Directory Access Protocol (LDAP) server and configure the java.security file.

Procedure

1. Add the following options to the configuration file:

Header	**Description**
`crl.ldap.host=`*`host_name`*	LDAP host name.
`crl.ldap.port=`*`port_number`*	LDAP server port number. You can specify up to 11 servers. Multiple LDAP hosts are used to ensure transparent failover in case of LDAP connection failure. It is expected that all LDAP servers are replicas and contain the same data. When the AMS Java interceptor successfully connects to an LDAP server, it does not attempt to download CRLs from the remaining servers provided. Java does not use `crl.ldap.user` and crl.ldaworldp.pass values. It does not use a user and password when connecting to an LDAP server. As a consequence, CRL attributes in LDAP must be world-readable.
`crl.cdp=on/off`	Use this option to check or use CRLDistributionPoints extensions in certificates.

2. Modify the `JRE/lib/security/java.security` file with the following properties:

Property Name	**Description**
`com.ibm.security.enableCRLDP`	This property takes the following values: `true`, `false`. If it is set to `true`, when doing certificate revocation check, CRLs are located using the URL from CRL distribution points extension of the certificate. If it is set to `false` or not set, checking CRL by using the CRL distribution points extension is disabled.
`ibm.security.certpath.ldap.cache.lifetime`	This property can be used to set the lifetime of entries in the memory cache of LDAP CertStore to a value in seconds. A value of 0 disables the cache; -1 means unlimited lifetime. If not set, the default lifetime is 30 seconds.

Property Name	Description
`com.ibm.security.enableAIAEXT`	This property takes the following values: `true`, `false`. If it is set to `true`, any Authority Information Access extensions that are found within the certificates of the certificate path being built are examined to determine whether they contain LDAP URIs. For each LDAP URI found, an LDAPCertStore object is created and added to the collection of CertStores that is used to locate other certificates that are required to build the certificate path. If it is set to `false` or not set, additional LDAPCertStore objects are not created.

z/OS *Enabling certificate revocation lists (CRLs) on z/OS*
Advanced Message Security supports Certificate Revocation List (CRL) checking of the digital certificates used to protect data messages

About this task

When enabled, Advanced Message Security will validate recipient certificates when messages are put to a privacy protected queue, and validate sender certificates when messages are retrieved from a protected queue (integrity or privacy). Validation in this case includes verification that relevant certificates are not registered in a relevant CRL.

Advanced Message Security uses IBM System SSL services to validate sender and recipient certificates. You can find detailed documentation regarding System SSL certificate validation in the z/OS Cryptographic Services System Secure Sockets Layer Programming manual.

To enabled CRL checking, you specify the location of a CRL configuration file via the CRLFILE DD in the started task JCL for the AMS address space. A sample CRL configuration file that can be customized is provided in *thlqual*.SCSQPROC(CSQ40CRL). Settings permitted in this file are as follows:

Table 105. Advanced Message Security CRL configuration variables

Variable	Valid values	Description
crl.ldap.host[.n]	*hostname -or- hostname:port*	The ipaddr/hostname of your LDAP server that hosts CRLs of your issuer certificates. If you do not specify a port number for your LDAP server, the port number specified by crl.ldap.port is used.
crl.ldap.port	*port*	The TCP/IP port number of your LDAP server.
crl.ldap.user	*ldap_user*	The LDAP user name to use when connecting to the LDAP server.
crl.ldap.pass	*ldap_password*	The LDAP password associated with the crl.ldap.user.

You can specify multiple LDAP server host names and ports as follows:

```
crl.ldap.host.1 = hostname -or hostname:port
crl.ldap.host.2 = hostname -or hostname:port
crl.ldap.host.3 = hostname -or hostname:port
```

You can specify up to 10 host names. If you do not specify a port number for your LDAP servers, the port number specified by crl.ldap.port is used. Each LDAP server must use the same crl.ldap.user/password combination for access.

When the CRLFILE DD is specified the configuration is loaded during initialization of the Advanced Message Security address space and CRL checking is enabled. If the CRLFILE DD is not specified, or the CRL configuration file is unavailable, or invalid, CRL checking is disabled.

AMS performs a CRL check using IBM System SSL certificate validation services as follows:

Table 106. Advanced Message Security CRL checks

Operation	**Quality of protection**	**Certificate(s) checked**
PUT	Privacy	Recipient(s)
GET	Integrity/Privacy	Sender

If a message operation fails a CRL check Advanced Message Security performs the following actions:

Table 107. Advanced Message Security CRL check failure behavior

Operation	**CRL check failure**
PUT	The message is not put to the target queue. A completion code of MQCC_FAILED and a reason code of MQRC_SECURITY_ERROR is returned to the application.
GET	The message is removed from the target queue and moved to the system protection error queue. A completion code of MQCC_FAILED and a reason code of MQRC_SECURITY_ERROR is returned to the application.

AMS for z/OS uses IBM System SSL services to validate certificates, which includes CRL and trust checking.

IBM MQ uses a security setting where certificate validation requires the LDAP server to be contactable, but does not require a CRL to be defined.

Note: It is the responsibility of administrators to ensure relevant LDAP services are available and to maintain CRL entries for relevant Certificate Authorities.

Setting up AMS password protection for configuration files

Storing keystore and private key passwords as plain text poses a security risk so Advanced Message Security provides a tool that can scramble those passwords using a user's key.

Before you begin

The `keystore.conf` file owner must ensure that only the file owner is entitled to read and write to the file. The passwords protection described in this topic, is only an additional measure of protection. Additionally, you should perform this procedure on a secure system.

Ensure you use the correct **`runamscred`** variant for the type of AMS client that is going to be reading the configuration file. If the AMS client is a:

- Java client, you should use the Java **`runamscred`** command, which is located in `<IBM MQ installation root>/java/bin`
- MQI client, you should use the MQI **`runmqascred`** command which is located in `<IBM MQ installation root>/bin`

Procedure

1. Edit the `keystore.conf` files to include all the required information, including the passwords that require protecting.

```
jceks.keystore = c:/Documents and Settings/Alice/AliceKeystore
jceks.certificate = AliceCert
jceks.encrypted = no
jceks.keystore_pass = passw0rd
jceks.key_pass = passw0rd
jceks.provider = IBMJCE
```

2. Place the encryption key to encrypt the passwords inside a file accessible to the user protecting the `keystore.conf` file.

   This key must be the same key that is going to be used by the AMS client later:

```
ThisIsAnExampleEncryptionKey
```

3. Run the **`runamscred`** command, to protect the `keystore.conf` file providing the encryption key file.

```
runamscred -f <location of keystore.conf> -sf <location of encryption keyfile>
```

4. Verify that the `keystore.conf` file has been protected and contains encrypted passwords.

Example

The following example shows what a protected `keystore.conf` file looks like:

```
jceks.keystore = c:/Documents and Settings/Alice/AliceKeystore
jceks.certificate = AliceCert
jceks.encrypted = yes
jceks.keystore_pass =
<AMS>1!62K/a4RinT+bks4RjFWx4A==!Vhi/RjIN2FH5qStUJ/0hsgKyn2IdMuhanemRRDrJq
HM=
jceks.key_pass =
<AMS>1!qmnxY++rsOUtZfDSgwcR1g==!VmWVREdVkNp1xYJstvuW64ph5vxxf7SPoqtsXxYh2
Tk=
jceks.provider = IBMJCE
```

Related information

runamscred: protect AMS keywords

z/OS Using certificates with AMS on z/OS

About this task

Advanced Message Security implements three levels of protection: integrity, confidentiality, and privacy.

With an integrity policy, messages are signed using the private key of the originator (the application doing the MQPUT). Integrity provides detection of message modification, but the message text itself is not encrypted.

With a confidentiality policy, the message is encrypted when it is put to the queue. The message is encrypted using a symmetric key and an algorithm specified in the relevant Advanced Message Security policy. The symmetric key itself is encrypted with the public key of each recipient (the application doing the MQGET). Public keys are associated with certificates stored in key rings.

With a privacy policy, messages are both signed and encrypted.

When a message that is protected with privacy is dequeued by a recipient application doing an MQGET, the message must be decrypted. Because it was encrypted using the recipient's public key, it must be decrypted using the recipient's private key found in a key ring.

z/OS *Use of SAF key rings with AMS on z/OS*

Advanced Message Security (AMS) makes use of z/OS SAF key ring services to define and manage the certificates needed for signing and encryption. Security products that are functionally equivalent to RACF may be used instead of RACF if they provide the same level of support.

Efficient use of key rings can reduce the administration needed to manage the certificates.

After a certificate is generated (or imported), it must be connected to a key ring to become accessible. The same certificate can be connected to more than one key ring.

Advanced Message Security uses two sets of key rings. One set consists of key rings owned by the individual user IDs that originate or receive messages. Each key ring contains the private key associated with the certificate of the owning user ID. The private key of each certificate is used to sign messages for integrity protected or privacy protected queues. It is also used to decrypt messages from privacy protected or confidentiality protected queues when receiving messages.

The other set is a single key ring owned by the AMS address space user. It contains the chain of signing CA certificates necessary to validate the certificates of the message originator and recipients.

When privacy or confidentiality protection is used, the key ring owned by the AMS address space user also contains the certificates of the message recipients. The public keys in these certificates are used to encrypt the symmetric key that was used to encrypt the message data when the message was put to the protected queue. When these messages are retrieved, the private key of relevant recipients is used to decrypt the symmetric key which is then used to decrypt the message data.

Advanced Message Security uses a key ring name of **drq.ams.keyring** when searching for certificates and private keys. This is the case for both the user and the AMS address space key rings.

For an illustration and further explanation of certificates and key ring, and their role in data protection, refer to Summary of the certificate-related operations.

The private key used for signing can have any label but must be connected as the default certificate. The private key or keys used for decryption can have any label, and must be connected to the key ring, but are no longer required to be connected as the default certificate.

Digital certificates and key rings are managed in RACF primarily by using the RACDCERT command.

For more information about certificates, labels, and the RACDCERT command, see the *z/OS: Security Server RACF Command Language Reference* and the *z/OS: Security Server RACF Security Administrator's Guide*.

z/OS *Replacing certificates*

When a certificate is renewed or replaced (for example, when the existing certificate is approaching its expiry date), it is not always possible to remove the protection from existing messages that are already on queues protected by confidentiality or privacy policies.

This can occur when the certificate was:

- Renewed with the same private key, and the reissued certificate has replaced the original certificate
- Re-keyed with a new private key and the RACDCERT ROLLOVER command has deleted the original private key

Messages will be decrypted, provided the necessary certificate is connected to the keyring of the user; it is no longer required to be connected as the default. This allows messages already on the queue, when the new certificate is connected, to be successfully decrypted.

The following example shows how a new certificate can be generated based on the existing certificate:

- A new certificate is created based on the existing certificate, with new public/private key pair.
- The new certificate is signed by the issuing authority.
- The public key of the old certificate is removed from the keyring of the AMS address space, and the public key of the new certificate is added.

- The new certificate and private key is added to the keyring of the user, in addition to the old certificate.

```
RACDCERT ID(user1) REKEY(LABEL('user1'))          -
         WITHLABEL('user1new')

RACDCERT GENREQ(LABEL('user1new')) ID(user1)      -
         DSN(output_data_set_name)

RACDCERT GENCERT(output_data_set_name) ID(user1)  -
         SIGNWITH(CERTAUTH LABEL('AMSCA'))

RACDCERT ID(user1) ALTER (LABEL('user1new'))      -
         TRUST

RACDCERT ID(WMQAMSD) REMOVE(ID(user1)             -
         LABEL('user1')                           -
         RING(drq.ams.keyring)  )

RACDCERT ID(WMQAMSD) CONNECT(ID(user1)            -
         LABEL('user1new') USAGE(SITE)            -
         RING(drq.ams.keyring)  )

RACDCERT ID(user1) CONNECT(ID(user1)              -
         LABEL('user1new') USAGE(PERSONAL)        -
         RING(drq.ams.keyring) DEFAULT )
```

For more information about certificates, labels, and the RACDCERT command, see the *z/OS: Security Server RACF Command Language Reference* and the *z/OS: Security Server RACF Security Administrator's Guide*.

z/OS *Authorizing access to the RACDCERT command for AMS on z/OS*

Authorization to use the RACDCERT command is a post-installation task that should have been completed by your z/OS system programmer. This task involves granting relevant permissions to the Advanced Message Security security administrator.

As a summary, these commands are needed to allow access to the RACF RACDCERT command:

```
RDEFINE FACILITY IRR.DIGTCERT.* UACC(NONE)
PERMIT IRR.DIGTCERT.* CLASS(FACILITY) ID( admin ) ACCESS(CONTROL)
SETROPTS RACLIST(FACILITY) REFRESH
```

In this example, *admin* specifies the user ID of your security administrator, or any user you want to use the RACDCERT command.

z/OS *Creating the certificates and key rings for AMS users on z/OS*

This section documents the steps required to create the certificates and key rings necessary for z/OS users of Advanced Message Security (AMS), using a RACF Certificate Authority (CA).

Resolving problems with certificates when using Advanced Message Security on z/OS

If you are having problems with certificates and missing entries in key stores you can enable a IBM Global Security Kit (GSKit) trace.

In the file referenced by the ENVARS DD in the AMS started task procedure, add:

```
GSK_TRACE_FILE=/u/... /gsktrace
GSK_TRACE=0xff
```

See Environment variables for more information.

For every access to the keystore, data is written to the trace file specified in `GSK_TRACE_FILE`.

To format the trace file use the command:

```
gsktrace inputtrace file > output_file
```

Scenario

A scenario of a sending application and a receiving application is used to explain the required steps.

In the examples that follow, `user1` is the originator of a message and `user2` is the recipient. The user ID of the Advanced Message Security address space is `WMQAMSD`.

All of the commands in the examples shown here are issued from ISPF option 6 by the administrative user ID `admin`.

z/OS *Defining a local Certificate Authority certificate for AMS on z/OS*

If you are using RACF as your CA, you must create a certificate authority certificate, if you have not already done so. The command shown here creates a certificate authority (or signer) certificate. This example creates a certificate called AMSCA to be used when creating subsequent certificates that reflect the identity of Advanced Message Security users and applications.

This command may be modified, specifically `SUBJECTSDN`, to reflect the naming structure and conventions used at your installation:

```
RACDCERT CERTAUTH GENCERT SUBJECTSDN(CN('AMSCA') O('ibm') C('us'))
KEYUSAGE(CERTSIGN) WITHLABEL('AMSCA')
```

Note: Certificates signed with this local certificate authority certificate show an issuer of CN=AMSCA,O=ibm,C=us when listed with the RACDCERT LIST command.

z/OS *Creating a digital certificate with a private key for AMS on z/OS*

A digital certificate with a private key must be generated for each Advanced Message Security user. In the example shown here, RACDCERT commands are used to generate certificates for user1 and user2, which are signed with the local CA certificate identified by the label AMSCA.

```
RACDCERT ID(user1) GENCERT SUBJECTSDN(CN('user1') O('ibm') C('us'))
WITHLABEL('user1') SIGNWITH(CERTAUTH LABEL('AMSCA'))
KEYUSAGE(HANDSHAKE DATAENCRYPT DOCSIGN)

RACDCERT ID(user2) GENCERT SUBJECTSDN(CN('user2') O('ibm') C('us'))
WITHLABEL('user2') SIGNWITH(CERTAUTH LABEL('AMSCA'))
KEYUSAGE(HANDSHAKE DATAENCRYPT DOCSIGN)

RACDCERT ID(user1) ALTER (LABEL('user1')) TRUST
RACDCERT ID(user2) ALTER (LABEL('user2')) TRUST
```

The RACDCERT ALTER command is required to add the TRUST attribute to the certificate. When a certificate is first created using this procedure, it has a different valid date range than the signing certificate. As a result, RACF marks it as NOTRUST, which means that the certificate is not to be used. Use the RACDCERT ALTER command to set the TRUST attribute.

The KEYUSAGE attributes HANDSHAKE, DATAENCRYPT and DOCSIGN must be specified for certificates used by Advanced Message Security.

Table 108. RACDCERT KEYUSAGE values and indicators

KEYUSAGE Value	**Indicators Set**
HANDSHAKE	digitalSignature and keyEncipherment
DATAENCRYPT	dataEncipherment
DOCSIGN	nonRepudiation

Table 108. RACDCERT KEYUSAGE values and indicators (continued)

KEYUSAGE Value	Indicators Set
CERTSIGN	keyCertSign and cRLSign

z/OS *Creating the RACF key rings for AMS on z/OS*

The commands shown here create a key ring for RACF-defined user IDs user1, user2, and the Advanced Message Security address space task user WMQAMSD. The key ring name is fixed by Advanced Message Security and must be coded as shown, without quotes. The name is case-sensitive.

```
RACDCERT ID(user1) ADDRING(drq.ams.keyring)
RACDCERT ID(user2) ADDRING(drq.ams.keyring)
RACDCERT ID(WMQAMSD) ADDRING(drq.ams.keyring)
```

z/OS *Connecting the certificates to the key rings for AMS on z/OS*

Connect the user and CA certificates to the key rings:

```
RACDCERT ID(WMQAMSD) CONNECT(CERTAUTH LABEL('AMSCA')
RING(drq.ams.keyring))
RACDCERT ID(user1) CONNECT(ID(user1) LABEL('user1')
RING(drq.ams.keyring) DEFAULT USAGE(PERSONAL))
RACDCERT ID(user2) CONNECT(ID(user2) LABEL('user2')
RING(drq.ams.keyring) DEFAULT USAGE(PERSONAL))
RACDCERT ID(WMQAMSD) CONNECT(ID(user2) LABEL('user2')
RING(drq.ams.keyring) USAGE(SITE))
```

Any certificates containing the private key or keys used for decryption must be connected to the key ring of the user, however they are no longer required to be connected as the default certificate.

The RACDCERT USAGE(SITE) attribute prevents the private key from being accessible in the key ring, while the RACDCERT USAGE(PERSONAL) attribute allows the private key to be used, if it exists. User2's certificate must be connected to the Advanced Message Security address space key ring because its public key is needed to encrypt messages as they are put to the queue. USAGE(SITE) limits exposure of user2's private key.

The CERTAUTH certificate with label AMSCA must be connected to the Advanced Message Security address space key ring because it was used to sign the certificate of user1, who is the message originator. It is used to validate user1's signing certificate.

z/OS *Key ring verification for AMS on z/OS*

The key ring should appear as shown here, after all commands have been entered:

```
RACDCERT ID(user1) LISTRING(drq.ams.keyring)
Digital ring information for user USER1:
Ring:>drq.ams.keyring<:

Certificate Label Name           Cert Owner   USAGE    DEFAULT
-------------------------------- ------------ -------- -------
user1                            ID(USER1)    PERSONAL YES

RACDCERT ID(user2) LISTRING(drq.ams.keyring)
Digital ring information for user USER2:
Ring:>drq.ams.keyring<:

Certificate Label Name           Cert Owner   USAGE    DEFAULT
-------------------------------- ------------ -------- -------
user2                            ID(USER2)    PERSONAL YES

RACDCERT ID(WMQAMSD) LISTRING(drq.ams.keyring)
Digital ring information for user WMQAMSD:
Ring:>drq.ams.keyring<:
```

```
Certificate Label Name           Cert Owner   USAGE    DEFAULT
-------------------------------- ------------ -------- -------
AMSCA                            CERTAUTH     CERTAUTH NO
user2                            ID(USER2)    SITE     NO
```

Listing the individual certificates also shows the ring association.

```
RACDCERT ID(user2) LIST(label('user2'))
Digital certificate information for user USER2:

***
Label: user2
Certificate ID: 2QfH8Pny9/LzpKKFmfFA
Status: TRUST
Start Date: 2010/05/03 22:59:53
End Date:  2011/05/04 22:59:52
Serial Number:>15<:
Issuer's Name:>OU=AMSCA.O=ibm.C=us<:
Subject's Name:>CN=user2.O=ibm.C=us<:
Key Usage: HANDSHAKE, DATAENCRYPT, DOCSIGN
Private Key Type: Non-ICSF
Private Key Size: 1024
Ring Associations:
Ring Owner: USER2
Ring:>drq.ams.keyring<:
Ring Owner: WMQAMSD
Ring:>drq.ams.keyring<:
```

To improve performance, the contents of the drq.ams.keyring associated with the AMS address space is cached for the life of the address space. Changes to that key ring do not become effective automatically. The administrator can refresh the cache by either:

- Stopping and restarting the queue manager.
- Using the z/OS MODIFY command:

  ```
  F qmgrAMSM,REFRESH KEYRING
  ```

Related tasks

Operating Advanced Message Security

z/OS *Summary of the certificate-related operations for AMS on z/OS*

Figure 35 on page 652 illustrates the relationships between sending and receiving applications and relevant certificates. The scenario illustrated involves remote queuing between two z/OS queue managers using a data-protection policy of privacy. In Figure 35 on page 652, "AMS" indicates " Advanced Message Security".

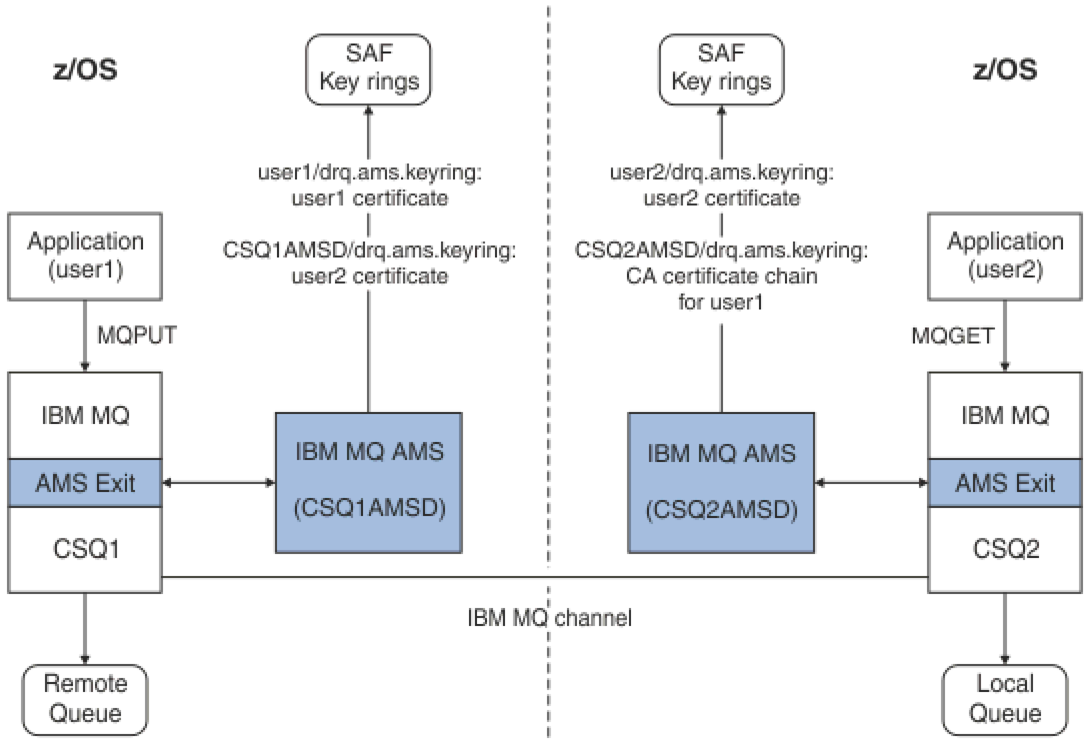


Figure 35. Application and certificate relationships

In this diagram, an application running as 'user1' puts a message to a remote queue managed by queue manager CSQ1, intended to be retrieved by an application running as 'user2' from a local queue managed by queue manager CSQ2. The diagram assumes an Advanced Message Security policy of privacy, which means the message is both signed and encrypted.

Advanced Message Security intercepts the message when a put occurs and uses user2's certificate (stored in the AMS address space user's key ring) to encrypt a symmetric key used to encrypt the message data.

Note that user2's certificate is connected to the AMS address space user key ring with option USAGE(SITE). This means the AMS address space user can access the certificate and public key, but not the private key.

On the receiving end, Advanced Message Security intercepts the get issued by user2, and uses user2's certificate to decrypt the symmetric key so that it can decrypt the message data. It then validates user1's signature using the CA certificate chain of user1's certificate stored in the AMS address space user's key ring.

Given this scenario, but with a data-protection policy of integrity, certificates for user2 would not be required.

To use Advanced Message Security to enqueue messages on IBM MQ-protected queues having a message protection policy of privacy or integrity, Advanced Message Security must have access to these data items:

- The X.509 V2 or V3 certificate and private key for the user enqueuing the message.
- The chain of certificates used to sign the digital certificates of all message signers.
- If the data protection policy is privacy, the X.509 V2 or V3 certificate of the intended recipients. The intended recipients are listed in the Advanced Message Security policy associated with the queue.

For processes and applications that run on z/OS, Advanced Message Security must have certificates in two places:

- In a SAF-managed key ring associated with the RACF identity of the sending application (the application that enqueues the protected message) or receiving application (if using privacy).

  The certificate that Advanced Message Security locates is the default certificate, and must include the private key. Advanced Message Security assumes the z/OS user identity of the sending application. That is, it acts as a surrogate, so it can access the user's private key.
- In a SAF-managed key ring associated with the AMS address space user.

  When sending messages protected with privacy, this key ring contains the public key certificates of the message recipients. When receiving messages, it contains the chain of Certificate Authority certificates needed to validate the message sender's signature.

The earlier examples shown have used RACF as the local CA. However, you may use another PKI provider (Certificate Authority) at your installation. If you intend to use another PKI product, remember that the private key and the certificate must be imported into a key ring associated with the z/OS RACF user IDs that originate IBM MQ messages protected by Advanced Message Security.

You can use the RACF RACDCERT command as the mechanism to generate certificate requests, which can be exported and sent to the PKI provider of your choice to be issued.

Here is a summary of the certificate-related steps:

1. Request the creation of a CA certificate, one in which RACF is the local CA. Omit this step if you are using another PKI provider.
2. Generate user certificates signed by the CA.
3. Create the key rings for the users and the Advanced Message Security AMS address space ID.
4. Connect the user certificate to the user key ring with the default attribute.
5. Connect the recipients certificates to the Advanced Message Security AMS address space user key ring using the usage(site) attribute (This step is necessary only for user certificates that will ultimately be the recipients of privacy-protected messages).
6. Connect the CA certificate chains for message senders to the Advanced Message Security AMS address space user key ring. (This step is necessary only for AMS tasks that will be verifying sender signatures.)

z/OS *Configuring a non-z/OS resident PKI for AMS*

Advanced Message Security for z/OS, uses X.509 V3 digital certificates in the protection-processing of messages placed on or received from IBM MQ queues. Advanced Message Security itself does not create or manage the life cycle of these certificates; that function is provided by a public key infrastructure (PKI). The examples in this publication that illustrate the use of certificates use z/OS Security Server RACF to fill certificate requests.

Whether or not a z/OS or non-z/OS resident PKI is used, AMS for z/OS uses only key rings that are managed by RACF or its equivalent. These key rings are based on Security Authorization Facility (SAF) and are the repository used by AMS for z/OS to retrieve certificates for originators and recipients of messages placed on or received from IBM MQ queues.

For messages that are originated from z/OS, which are protected by either integrity or encryption policy, the certificate and private key of the originating user ID must be stored in an SAF-managed key ring that is associated with the z/OS user ID of the message originator.

RACF includes the capability for importing certificates and private keys into RACF-managed key rings. See the z/OS Security Server RACF publications for the details and examples of how to load certificates to RACF managed key rings.

If your installation is using one of the supported PKI products, refer to the publications that accompany the product for information on how to deploy it.

Administering Advanced Message Security security policies

Advanced Message Security uses security policies to specify the cryptographic encryption and signature algorithms for encrypting and authenticating messages that flow through the queues.

Security policies overview for AMS

Advanced Message Security security policies are conceptual objects that describe the way a message is cryptographically encrypted and signed.

For details of the security policy attributes, see the following subtopics:

Related concepts

"Quality of protection in AMS" on page 658
Advanced Message Security data-protection policies imply a quality of protection (QOP).

"Security policy attributes in AMS" on page 657
You can use Advanced Message Security to select a particular algorithm or method to protect the data.

Policy names in AMS

The policy name is a unique name that identifies a specific Advanced Message Security policy and the queue to which it applies.

The policy name must be the same as the queue name to which it applies. There is a one-to-one mapping between an Advanced Message Security (AMS) policy and a queue.

By creating a policy with the same name as a queue, you activate the policy for that queue. Queues without matching policy names are not protected by AMS.

The scope of the policy is relevant to the local queue manager and its queues. Remote queue managers must have their own locally-defined policies for the queues they manage.

Signature algorithm in AMS

The signature algorithm indicates the algorithm that should be used when signing data messages.

Valid values include:

- `MD5`
- `SHA-1`
- `SHA-2` family:
  - SHA256
  - SHA384 (minimum key length acceptable - 768 bits)
  - SHA512 (minimum key length acceptable - 768 bits)

A policy that does not specify a signature algorithm, or specifies an algorithm of `NONE`, implies that messages placed on the queue associated with the policy are not signed.

Note: The quality of protection used for the message put and get functions must match. If there is a policy quality of protection mismatch between the queue and the message in the queue, the message is not accepted and is sent to the error handling queue. This rule applies for both local and remote queues.

Encryption algorithm in AMS

The encryption algorithm indicates the algorithm that should be used when encrypting data messages placed on the queue associated with the policy.

Valid values include:

- Deprecated RC2
- Deprecated DES
- Deprecated 3DES

- AES128
- AES256

A policy that does not specify an encryption algorithm or specifies an algorithm of NONE implies that messages placed on the queue associated with the policy are not encrypted.

Note that a policy that specifies an encryption algorithm other than NONE must also specify at least one Recipient DN and a signature algorithm because Advanced Message Security encrypted messages are also signed.

Important: The quality of protection used for the message put and get functions must match. If there is a policy quality of protection mismatch between the queue and the message in the queue, the message is not accepted and is sent to the error handling queue. This rule applies for both local and remote queues.

Toleration in AMS

The toleration attribute indicates whether Advanced Message Security can accept messages with no security policy specified.

When retrieving a message from a queue with a policy to encrypt messages, if the message is not encrypted, it is returned to the calling application. Valid values include:

0
No (**default**).

1
Yes.

A policy that does not specify a toleration value or specifies 0, implies that messages placed on the queue associated with the policy must match the policy rules.

Toleration is optional and exists to facilitate configuration roll-out, where policies were applied to queues but those queues already contain messages that do not have a security policy specified.

Sender distinguished names in AMS

The sender distinguished names (DNs) identify users who are authorized to place messages on a queue. A sender uses their certificate to sign a message, prior to placing the message on a queue.

Advanced Message Security (AMS) does not check whether a message has been placed on a data-protected queue by a valid user until the message is retrieved. At this time, if the policy stipulates one or more valid senders, and the user that placed the message on the queue is not in the list of valid senders, AMS returns an error to the receiving application, and places the message on the AMS error queue.

A policy can have 0 or more sender DNs specified. If no sender DNs are specified for the policy, any sender can put data-protected messages to the queue providing the sender's certificate is trusted. A sender's certificate is trusted by adding the public certificate to a keystore available to the receiving application.

Sender distinguished names have the following form:

```
CN=Common Name,O=Organization,C=Country
```

Important:

- All DN Component names must be in uppercase. All component name identifiers in the DN must be specified in the order shown in the following table:

Component name	Value
CN	The common name for the object of this DN, such as a full name or the intended purpose of a device.
OU	The unit within the organization with which the object of the DN is affiliated, such as a corporate division or a product name.

Component name	Value
O	The organization with which the object of the DN is affiliated, such as a corporation.
L	The locality (city or municipality) where the object of the DN is located.
ST	The state or province name where the object of the DN is located.
C	The country where the object of the distinguished name (DN) is located.

- If one or more sender DNs are specified for the policy, only those users can put messages to the queue associated with the policy.
- Sender DNs, when specified, must match exactly the DN contained in the digital certificate associated with user putting the message.
- AMS supports DNs with values only from Latin-1 character set. To create DNs with characters of the set, you must first create a certificate with a DN that is created in UTF-8 coding using AIX and Linux with UTF-8 coding turned on. Then you must create a policy from a Linux or AIX platform with UTF-8 coding turned on, or use the AMS plug-in to IBM MQ.
- The method used by AMS, to convert the name of the sender from x.509 format to DN format, always uses ST= for the State or Province value.
- The following special characters need escape characters:

```
      , (comma)
      + (plus)
      " (double quote)
      \ (backslash)
      < (less than)
      > (greater than)
      ; (semicolon)
```

- If the distinguished name contains embedded blanks, you should enclose the DN in double quotation marks.

Related concepts

"Recipient distinguished names in AMS" on page 656
The recipient distinguished names (DN) identify users who are authorized to retrieve messages from a queue.

Recipient distinguished names in AMS

The recipient distinguished names (DN) identify users who are authorized to retrieve messages from a queue.

A policy can have zero or more recipient DNs specified. Recipient distinguished names have the following form:

```
CN=Common Name,O=Organization,C=Country
```

Important:

- All DN Component names must be in uppercase. All component name identifiers in the DN must be specified in the order shown in the following table:

Component name	Value
CN	The common name for the object of this DN, such as a full name or the intended purpose of a device.

Component name	Value
OU	The unit within the organization with which the object of the DN is affiliated, such as a corporate division or a product name.
O	The organization with which the object of the DN is affiliated, such as a corporation.
L	The locality (city or municipality) where the object of the DN is located.
ST	The state or province name where the object of the DN is located.
C	The country where the object of the distinguished name (DN) is located.

- If no recipient DNs are specified for the policy, any user can get messages from the queue associated with the policy.
- If one or more recipient DNs are specified for the policy, only those users can get messages from the queue associated with the policy.
- Recipient DNs, when specified, must match exactly the DN contained in the digital certificate associated with user getting the message.
- Advanced Message Security supports DNs with values only from Latin-1 character set. To create DNs with characters of the set, you must first create a certificate with a DN that is created in UTF-8 coding using AIX or Linux with UTF-8 coding turned on. Then you must create a policy from a Linux or AIX platform with UTF-8 coding turned on or use the Advanced Message Security plug-in to IBM MQ.

Related concepts

"Sender distinguished names in AMS" on page 655
The sender distinguished names (DNs) identify users who are authorized to place messages on a queue. A sender uses their certificate to sign a message, prior to placing the message on a queue.

Security policy attributes in AMS

You can use Advanced Message Security to select a particular algorithm or method to protect the data.

A security policy is a conceptual object that describes the way a message is cryptographically encrypted and signed.

Table 109. Security policy attributes in AMS

Attributes	Description
Policy name	Unique name of the policy for a queue manager.
Signature algorithm	Cryptographic algorithm that is used to sign messages before sending.
Encryption algorithm	Cryptographic algorithm that is used to encrypt messages before sending.
Recipient list	List of certificate distinguished names (DNs) of potential receivers of a message.
Signature DN checklist	List of signature DNs to be validated during message retrieval.

In Advanced Message Security, messages are encrypted with a symmetric key, and the symmetric key is encrypted with the public keys of the recipients. Public keys are encrypted with the RSA algorithm, with keys of an effective length up to 2048 bits. The actual asymmetric key encryption depends on the certificate key length.

The supported symmetric-key algorithms are as follows:

- Deprecated RC2
- Deprecated DES
- Deprecated 3DES
- AES128
- AES256

Advanced Message Security also supports the following cryptographic hash functions:

- Deprecated MD5
- Deprecated SHA-1
- SHA-2 family:
  - SHA256
  - SHA384 (minimum key length acceptable - 768 bits)
  - SHA512 (minimum key length acceptable - 768 bits)

Note: The quality of protection used for the message put and get functions must match. If there is a policy quality of protection mismatch between the queue and the message in the queue, the message is not accepted and is sent to the error handling queue. This rule applies for both local and remote queues.

Quality of protection in AMS

Advanced Message Security data-protection policies imply a quality of protection (QOP).

The three quality of protection levels in Advanced Message Security are supplemented by a fourth level in IBM MQ 9.0 and later, and all depend on cryptographic algorithms that are used to sign and encrypt the message:

- Privacy - messages placed on the queue must be signed and encrypted.
- Integrity - messages placed on the queue must be signed by the sender.
- Confidentiality - messages placed on the queue must be encrypted. For more information, see "Qualities of protection available with AMS" on page 584
- None - no data protection is applicable.

A policy that stipulates that messages must be signed when placed on a queue has a QOP of INTEGRITY. A QOP of INTEGRITY means that a policy stipulates a signature algorithm, but does not stipulate an encryption algorithm. Integrity-protected messages are also referred to as "SIGNED".

A policy that stipulates that messages must be signed and encrypted when placed on a queue has a QOP of PRIVACY. A QOP of PRIVACY means that when a policy stipulates a signature algorithm and an encryption algorithm. Privacy-protected messages are also referred to as "SEALED".

A policy that stipulates that messages must be encrypted when placed on a queue has a QOP of CONFIDENTIALITY. A QOP of CONFIDENTIALITY means that a policy stipulates an encryption algorithm.

A policy that does not stipulate a signature algorithm or an encryption algorithm has a QOP of NONE. Advanced Message Security provides no data-protection for queues that have a policy with a QOP of NONE.

Managing security policies in AMS

A security policy is a conceptual object that describes the way a message is cryptographically encrypted and signed.

The location from which all administrative tasks related to security policies are run depends on which platform you are using.

- ALW On AIX, Linux, and Windows, you use the DELETE POLICY, DISPLAY POLICY, and SET POLICY (or equivalent PCF) commands to manage your security policies.
  - Linux AIX On AIX and Linux, administrative tasks can be run from *MQ_INSTALLATION_PATH*/bin.
  - Windows On Windows platforms, administrative tasks can be run from any location as the PATH environment variable is updated at the installation.
- IBM i On IBM i, the DSPMQMSPL, SETMQMSPL, and WRKMQMSPL commands are installed into the QSYS system library for the primary language of the system when IBM MQ is installed.

  Additional national language versions get installed into QSYS29xx libraries according to the language feature load. For example, a machine with US English as the primary language and Korean as the secondary language has the US English commands installed into QSYS and the Korean secondary language load in QSYS2962 as 2962 is the language load for Korean.
- z/OS On z/OS, the administrative commands are run using the message security policy utility (CSQ0UTIL). When policies are created, modified or deleted on z/OS, the changes are not recognized by Advanced Message Security until the queue manager is stopped and restarted, or the z/OS MODIFY command is used to refresh the Advanced Message Security policy configuration. For example:

```
F <qmgr ssid>AMSM,REFRESH POLICY
```

Related tasks

"Creating security policies in AMS" on page 659
Security policies define the way in which a message is protected when the message is put, or how a message must have been protected when a message is received.

"Changing security policies in AMS" on page 660
You can use Advanced Message Security to alter details of security policies that you have already defined.

"Displaying and dumping security policies in AMS" on page 661
Use the **dspmqspl** command to display a list of all security policies or details of a named policy depending on the command-line parameters you supply.

"Removing security policies in AMS" on page 662
To remove security policies in Advanced Message Security, you must use the `setmqspl` command.

Operating Advanced Message Security

Related reference

The message security policy utility (CSQ0UTIL)

Creating security policies in AMS

Security policies define the way in which a message is protected when the message is put, or how a message must have been protected when a message is received.

Before you begin

There are some entry conditions which must be met when creating security policies:

- The queue manager must be running.
- The name of a security policy must follow Rules for naming IBM MQ objects.
- You must have the necessary authority to connect to the queue manager and create a security policy:
  - z/OS On z/OS, grant the authorities documented in The message security policy utility (CSQ0UTIL).
  - Multi On Multiplatforms, you must grant the necessary `+connect`, `+inq` and `+chg` authorities using the **`setmqaut`** command.

For more information about configuring security see "Setting up security" on page 129.

- z/OS On z/OS, ensure the required system objects have been defined according to the definitions in CSQ4INSM.

Example

Here is an example of creating a policy on queue manager QMGR. The policy specifies that messages be signed using the SHA256 algorithm and encrypted using the AES256 algorithm for certificates with DN: CN=joe,O=IBM,C=US and DN: CN=jane,O=IBM,C=US. This policy is attached to MY.QUEUE:

```
setmqspl -m QMGR -p MY.QUEUE -s SHA256 -e AES256 -r CN=joe,O=IBM,C=US -r CN=jane,O=IBM,C=US
```

Here is an example of creating policy on the queue manager QMGR. The policy specifies that messages be encrypted using the 3DES algorithm for certificates with DNs: CN=john,O=IBM,C=US and CN=jeff,O=IBM,C=US and signed with the SHA256 algorithm for certificate with DN: CN=phil,O=IBM,C=US

```
setmqspl -m QMGR -p MY.OTHER.QUEUE -s SHA256 -e 3DES -r CN=john,O=IBM,C=US -r
CN=jeff,O=IBM,C=US -a CN=phil,O=IBM,C=US
```

Note:

- The quality of protection being used for the message put and get must match. If the policy quality of protection that is defined for the message is weaker than that defined for a queue, the message is sent to the error handling queue. This policy is valid for both local and remote queues.

Related reference

Complete list of setmqspl command attributes

Changing security policies in AMS

You can use Advanced Message Security to alter details of security policies that you have already defined.

Before you begin

- The queue manager on which you want to operate must be running.
- You must have the necessary authority to connect to the queue manager and create a security policy.
  - z/OS On z/OS, grant the authorities documented in The message security policy utility (CSQ0UTIL).
  - Multi On Multiplatforms, you must grant the necessary `+connect`, `+inq` and `+chg` authorities using the **setmqaut** command.

  For more information about configuring security see "Setting up security" on page 129.

About this task

To change security policies, apply the `setmqspl` command to an already existing policy providing new attributes.

Example

Here is an example of creating a policy named MYQUEUE on a queue manager named QMGR, specifying that messages are to be encrypted using the 3DES algorithm for authors (**-a**) having certificates with Distinguished Name (DN) of CN=alice,O=IBM,C=US and signed with the SHA256 algorithm for recipients (**-r**) having certificates with DN of CN=jeff,O=IBM,C=US.

```
setmqspl -m QMGR -p MYQUEUE -e 3DES -s SHA256 -a CN=jeff,O=IBM,C=US -r CN=alice,O=IBM,C=US
```

To alter this policy, issue the `setmqspl` command with all attributes from the example changing only the values you want to modify. In this example, previously created policy is attached to a new queue and its encryption algorithm is changed to AES256:

```
setmqspl -m QMGR -p MYQUEUE -e AES256 -s SHA256 -a CN=jeff,O=IBM,C=US -r CN=alice,O=IBM,C=US
```

Related reference

setmqspl (set security policy)

Displaying and dumping security policies in AMS

Use the **`dspmqspl`** command to display a list of all security policies or details of a named policy depending on the command-line parameters you supply.

Before you begin

- To display security policies details, the queue manager must exist, and be running.
- You must have the necessary authority to connect to the queue manager and create a security policy.
  - z/OS On z/OS, grant the authorities documented in The message security policy utility (CSQ0UTIL).
  - Multi On Multiplatforms, you must grant the necessary `+connect`, `+inq` and `+chg` authorities using the **`setmqaut`** command.

  For more information about configuring security see "Setting up security" on page 129.

About this task

Here is the list of **`dspmqspl`** command flags:

*Table 110. **`dspmqspl`** command flags.*

Command flag	Explanation
`-m`	Queue manager name (mandatory).
`-p`	Policy name.
`-export`	Adding this flag generates output which can easily be applied to a different queue manager.

Example

The following example shows how to create two security policies for `venus.queue.manager`:

```
setmqspl -m venus.queue.manager -p AMS_POL_04_ONE -s sha256 -a "CN=signer1,O=IBM,C=US" -e NONE
setmqspl -m venus.queue.manager -p AMS_POL_06_THREE -s sha256 -a "CN=another signer,O=IBM,C=US"
-e NONE
```

This example shows a command that displays details of all policies defined for `venus.queue.manager` and the output it produces:

```
dspmqspl -m venus.queue.manager

Policy Details:
Policy name: AMS_POL_04_ONE
Quality of protection: INTEGRITY
Signature algorithm: SHA256
Encryption algorithm: NONE
Signer DNs:
  CN=signer1,O=IBM,C=US
Recipient DNs: -
Toleration: 0
```

```
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Policy Details:
Policy name: AMS_POL_06_THREE
Quality of protection: INTEGRITY
Signature algorithm: SHA256
Encryption algorithm: NONE
Signer DNs:
  CN=another signer,O=IBM,C=US
Recipient DNs: -
Toleration: 0
```

This example shows a command that displays details of a selected security policy defined for `venus.queue.manager` and the output it produces:

```
dspmqspl -m venus.queue.manager -p AMS_POL_06_THREE

Policy Details:
Policy name: AMS_POL_06_THREE
Quality of protection: INTEGRITY
Signature algorithm: SHA256
Encryption algorithm: NONE
Signer DNs:
  CN=another signer,O=IBM,C=US
Recipient DNs: -
Toleration: 0
```

In the next example, first, we create a security policy and then, we export the policy using the **`-export`** flag:

```
setmqspl -m venus.queue.manager -p AMS_POL_04_ONE -s SHA256 -a "CN=signer1,O=IBM,C=US" -e NONE

dspmqspl -m venus.queue.manager -export
```

z/OS On z/OS, the exported policy information is written by CSQ0UTIL to the EXPORT DD.

Multi On Multiplatforms, redirect the output to a file, for example:

```
dspmqspl -m venus.queue.manager -export > policies.[bat|sh]
```

To import a security policy:

- Linux AIX On AIX and Linux:
  1. Log on as a user that belongs to the mqm IBM MQ administration group.
  2. Issue `. policies.sh`.
- Windows On Windows, run `policies.bat`.
- z/OS On z/OS use the CSQ0UTIL utility, specifying to SYSIN the data set containing the exported policy information.

Related reference

Complete list of dspmqspl command attributes

Removing security policies in AMS

To remove security policies in Advanced Message Security, you must use the `setmqspl` command.

Before you begin

There are some entry conditions which must be met when managing security policies:

- The queue manager must be running.
- You must have the necessary authority to connect to the queue manager and create a security policy.

- z/OS On z/OS, grant the authorities documented in The message security policy utility (CSQ0UTIL).
- Multi On Multiplatforms, you must grant the necessary `+connect`, `+inq` and `+chg` authorities using the **`setmqaut`** command.

For more information about configuring security see “Setting up security” on page 129.

About this task

Use the **`setmqspl`** command with the **`-remove`** option.

Example

Here is an example of removing a policy:

```
setmqspl -m QMGR -remove -p MY.OTHER.QUEUE
```

Related reference

Complete list of setmqspl command attributes

System queue protection in AMS

System queues enable communication between IBM MQ and its ancillary applications. Whenever a queue manager is created, a system queue is also created to store IBM MQ internal messages and data. You can protect system queues with Advanced Message Security so that only authorized users can access or decrypt them.

System queue protection follows the same pattern as the protection of regular queues. See “Creating security policies in AMS” on page 659.

Windows To use system queue protection on Windows, copy the `keystore.conf` file to the following directory:

```
c:\Documents and Settings\Default User\.mqs\keystore.conf
```

z/OS On z/OS, to provide protection for `SYSTEM.ADMIN.COMMAND.QUEUE`, the command server must have access to the `keystore` and the `keystore.conf`, which contain keys and a configuration so that the command server can access keys and certificates. All changes made to the security policy of `SYSTEM.ADMIN.COMMAND.QUEUE` require the restart of the command server.

All messages that are sent and received from the command queue are signed or signed and encrypted depending on policy settings. If an administrator defines authorized signers, command messages that do not pass the signer Distinguished Name (DN) check are not executed by the command server and are not routed to the Advanced Message Security error handling queue. Messages that are sent as replies to IBM MQ Explorer temporary dynamic queues are not protected by AMS.

Security policies do not have an effect on the following SYSTEM queues:

- SYSTEM.ADMIN.ACCOUNTING.QUEUE
- SYSTEM.ADMIN.ACTIVITY.QUEUE
- SYSTEM.ADMIN.CHANNEL.EVENT
- SYSTEM.ADMIN.COMMAND.EVENT
- z/OS SYSTEM.ADMIN.COMMAND.QUEUE
- SYSTEM.ADMIN.CONFIG.EVENT
- SYSTEM.ADMIN.LOGGER.EVENT
- SYSTEM.ADMIN.PERFM.EVENT

- SYSTEM.ADMIN.PUBSUB.EVENT
- SYSTEM.ADMIN.QMGR.EVENT
- SYSTEM.ADMIN.STATISTICS.QUEUE
- SYSTEM.ADMIN.TRACE.ROUTE.QUEUE
- SYSTEM.AUTH.DATA.QUEUE
- SYSTEM.BROKER.ADMIN.STREAM
- z/OS SYSTEM.BROKER.CLIENTS.DATA
- SYSTEM.BROKER.CONTROL.QUEUE
- SYSTEM.BROKER.DEFAULT.STREAM
- SYSTEM.BROKER.INTER.BROKER.COMMUNICATIONS
- z/OS SYSTEM.BROKER.SUBSCRIPTIONS.DATA
- SYSTEM.CHANNEL.INITQ
- SYSTEM.CHANNEL.SYNCQ
- z/OS SYSTEM.CHLAUTH.DATA.QUEUE
- SYSTEM.CICS.INITIATION.QUEUE
- SYSTEM.CLUSTER.COMMAND.QUEUE
- SYSTEM.CLUSTER.HISTORY.QUEUE
- SYSTEM.CLUSTER.REPOSITORY.QUEUE
- SYSTEM.CLUSTER.TRANSMIT.QUEUE
- z/OS SYSTEM.COMMAND.INPUT
- z/OS SYSTEM.DDELAY.LOCAL.QUEUE
- SYSTEM.DEAD.LETTER.QUEUE
- SYSTEM.DURABLE.SUBSCRIBER.QUEUE
- SYSTEM.HIERARCHY.STATE
- SYSTEM.INTER.QMGR.CONTROL
- SYSTEM.INTER.QMGR.FANREQ
- SYSTEM.INTER.QMGR.PUBS
- SYSTEM.INTERNAL.REPLY.QUEUE
- z/OS SYSTEM.JMS.PS.STATUS.QUEUE
- z/OS SYSTEM.JMS.REPORT.QUEUE
- SYSTEM.PENDING.DATA.QUEUE
- SYSTEM.PROTECTION.ERROR.QUEUE
- SYSTEM.PROTECTION.POLICY.QUEUE
- z/OS SYSTEM.QSG.CHANNEL.SYNCQ
- z/OS SYSTEM.QSG.TRANSMIT.QUEUE
- z/OS SYSTEM.QSG.UR.RESOLUTION.QUEUE
- SYSTEM.RETAINED.PUB.QUEUE
- z/OS SYSTEM.RETAINED.PUB.QUEUE
- SYSTEM.SELECTION.EVALUATION.QUEUE
- SYSTEM.SELECTION.VALIDATION.QUEUE

Multi Streaming queues and AMS

It is possible to stream duplicate Advanced Message Security (AMS) protected messages.

If a queue has an AMS policy defined that causes messages put to that queue to be signed and/or encrypted, you can also configure the **STREAMQ** attribute of the queue to put a copy of each protected message to a second queue. The duplicate, streamed message is signed and/or encrypted using the same policy that has been configured for the original queue.

In the following example you are configuring two queues, QUEUE1 and QUEUE2. QUEUE1 has its **STREAMQ** attribute configured to put streamed messages to QUEUE2:

```
DEFINE QLOCAL(QUEUE2)
```

```
DEFINE QLOCAL(QUEUE1) STREAMQ(QUEUE2)
```

AMS protected messages are being put to QUEUE1 by a user with the certificate `CN=bob,O=IBM,C=GB`.

An application with certificate `CN=alice,O=IBM,C=GB` is going to consume the messages from QUEUE1. A separate application with certificate `CN=fred,O=IBM,C=GB` is going to consume the messages from QUEUE2.

QUEUE1 has the following AMS privacy policy applied to it:

```
SET POLICY(QUEUE1) SIGNALG(SHA256) SIGNER('CN=bob,O=IBM,C=GB') ENCALG(AES256)
RECIP('CN=alice,O=IBM,C=GB') RECIP('CN=fred,O=IBM,C=GB') ACTION(ADD)
```

If an encryption algorithm has been configured in the policy for QUEUE1, the recipients listed in the policy must include both the recipients of the original messages from QUEUE1, and the recipients who are going to consume duplicate messages from QUEUE2.

When the application attempts to consume messages from QUEUE2 it performs integrity checks, and/or decrypts the message based on the policy that has been set on QUEUE2. If an application wants to consume streamed messages from QUEUE2, you must set a suitable policy on QUEUE2 that allows the messages to be checked for integrity and decrypted correctly.

In particular, the signing algorithm, the signer, and the encryption algorithm must be the same as the policy applied to QUEUE1. The policy recipients for QUEUE2 must include the identity of the recipient consuming the message from QUEUE2.

Note: It is not necessary for the policy applied to QUEUE2 to list all of the recipients named in the policy set on QUEUE1.

For example, the following policy could be set on QUEUE2 to allow an application with the certificate distinguished name `CN=fred,O=IBM,C=GB` to read AMS-protected messages from it:

```
SET POLICY(QUEUE2) SIGNALG(SHA256) SIGNER('CN=bob,O=IBM,C=GB') ENCALG(AES256)
RECIP('CN=fred,O=IBM,C=GB') ACTION(ADD)
```

Related concepts

Streaming queues

Granting OAM permissions in AMS

File permissions authorize all users to execute `setmqspl` and `dspmqspl` commands. However, Advanced Message Security relies on the Object Authority Manager (OAM) and every attempt to execute these commands by a user who does not belong to the mqm group, which is the IBM MQ administration group, or does not have permissions to read security policy settings that are granted, results in an error.

Procedure

To grant necessary permissions to a user, run:

```
setmqaut -m SOME.QUEUE.MANAGER -t qmgr -p SOME.USER +connect +inq
setmqaut -m SOME.QUEUE.MANAGER -t queue -n SYSTEM.PROTECTION.POLICY.QUEUE -p SOME.USER +browse
```

```
+put
setmqaut -m SOME.QUEUE.MANAGER -t queue -n SYSTEM.PROTECTION.ERROR.QUEUE -p SOME.USER +put
```

Note: You only need to set these OAM authorities if you intend to connect clients, to the queue manager, using Advanced Message Security 7.0.1.

Attention: Browse authority to the SYSTEM.PROTECTION.POLICY.QUEUE is not mandatory in all situations. IBM MQ optimizes performance by caching policies so that you do not have to browse records for policy details on the SYSTEM.PROTECTION.POLICY.QUEUE in all cases.

IBM MQ does not cache all the policies available. If there are high number of policies, IBM MQ caches a limited number of policies. So, if the queue manager has a low number of policies defined, there is no need to provide the browse option to the SYSTEM.PROTECTION.POLICY.QUEUE.

However, you should give browse authority to this queue, in case there is a high number of policies defined, or if you are using old clients. The SYSTEM.PROTECTION.ERROR.QUEUE is used to put error messages generated by the AMS code. The put authority against this queue is checked only when you attempt to put an error message to the queue. Your put authority against the queue is not checked when you attempt to put or get message from an AMS protected queue.

Granting security permissions in AMS

When using command resource security you must set up permissions to allow Advanced Message Security to function. This topic uses RACF commands in the examples. If your enterprise uses a different external security manager (ESM) you must use the equivalent commands for that ESM.

There are three aspects to granting security permissions:

- "The AMSM address space" on page 666
- "CSQ0UTIL" on page 667
- "Using queues that have an Advanced Message Security policy defined" on page 667

Notes: The example commands use the following variables.

1. *QMgrName* - the name of the queue manager.

   z/OS On z/OS, this value can also be the name of a queue sharing group.
2. *username* - this can be a group name.
3. The examples show the MQQUEUE class. z/OS This can also be MXQUEUE, GMQQUEUE or GMXQUEUE. See "Profiles for queue security" on page 198 for further information.

Furthermore, if the profile already exists, you do not require the RDEFINE command.

The AMSM address space

You need to issue some IBM MQ security to the user name that the Advanced Message Security address space runs under.

- For batch connection to the queue manager, issue

```
RDEFINE MQCONN QMgrName.BATCH UACC(NONE)
            PERMIT QMgrName.BATCH CLASS(MQCONN) ID(username) ACCESS(READ)
```

- For access to the SYSTEM.PROTECTION.POLICY.QUEUE, issue:

```
RDEFINE MQQUEUE QMgrName.SYSTEM.PROTECTION.POLICY.QUEUE UACC(NONE)
            PERMIT QMgrName.SYSTEM.PROTECTION.POLICY.QUEUE CLASS(MQQUEUE)
ID(username) ACCESS(READ)
```

CSQ0UTIL

The utility that allows users to run the **setmqspl** and **dspmqspl** commands requires the following permissions, where the user name is the job user ID:

- For batch connection to the queue manager, issue:

```
  RDEFINE MQCONN QMgrName.BATCH UACC(NONE)
           PERMIT QMgrName.BATCH CLASS(MQCONN) ID(username) ACCESS(READ)
```

- For access to the SYSTEM.PROTECTION.POLICY.QUEUE, required for the **setmqpol** command, issue:

```
  RDEFINE MQQUEUE QMgrName.SYSTEM.PROTECTION.POLICY.QUEUE UACC(NONE)
            PERMIT QMgrName.SYSTEM.PROTECTION.POLICY.QUEUE CLASS(MQQUEUE)
ID(username) ACCESS(ALTER)
```

- For access to the SYSTEM.PROTECTION.POLICY.QUEUE, required for the **dspmqpol** command, issue:

```
  RDEFINE MQQUEUE QMgrName.SYSTEM.PROTECTION.POLICY.QUEUE UACC(NONE)
              PERMIT QMgrName.SYSTEM.PROTECTION.POLICY.QUEUE CLASS(MQQUEUE)
ID(username) ACCESS(READ)
```

Using queues that have an Advanced Message Security policy defined

When an application does any work with queues that have a policy defined on them, that application requires additional permissions to allow Advanced Message Security to protect messages.

The application requires:

- Read access to the SYSTEM.PROTECTION.POLICY.QUEUE. Do this by issuing:

```
RDEFINE MQQUEUE QMgrName.SYSTEM.PROTECTION.POLICY.QUEUE UACC(NONE)
          PERMIT QMgrName.SYSTEM.PROTECTION.POLICY.QUEUE CLASS(MQQUEUE)
ID(username) ACCESS(READ)
```

- Put access to the SYSTEM.PROTECTION.ERROR.QUEUE. Do this by issuing:

```
RDEFINE MQQUEUE QMgrName.SYSTEM.PROTECTION.ERROR.QUEUE UACC(NONE)
            PERMIT QMgrName.SYSTEM.PROTECTION.ERROR.QUEUE CLASS(MQQUEUE)
ID(username) ACCESS(READ)
```

IBM i Setting up certificates and the keystore configuration file for AMS on IBM i

Your first task when setting up Advanced Message Security protection is to create a certificate, and associate that with your environment. The association is configured through a file held in the integrated filesystem (IFS).

Procedure

1. To create a self-signed certificate using the OpenSSL tooling shipped with IBM i, issue the following command from QShell:

   ```
   /QOpenSys/usr/bin/openssl req -x509 -newkey rsa:2048 -keyout
   $HOME/private.pem -out $HOME/mycert.pem -nodes -days 365
   ```

   The command prompts for various distinguished name attributes for a new self-signed certificate, including:

   - Common Name (CN=)
   - Organization (O=)
   - Country (C=)

This creates an unencrypted private key and a matching certificate, both in PEM (Privacy Enhanced Mail) format.

For simplicity, just enter values for common name, organization, and country. These attributes and values are important when creating a policy.

Additional prompts and attributes can be customized by specifying a custom openssl configuration file on the command line with the **`-config`** parameter. Refer to OpenSSL documentation for more details on the configuration file syntax.

For example, the following command adds additional X.509 v3 certificate extensions:

```
/QOpenSys/usr/bin/openssl req -x509 -newkey rsa:2048
-keyout $HOME/private.pem -out $HOME/mycert.pem -nodes -days 365 -config myconfig.cnf
```

where myconfig.cnf is an ASCII stream file that contains the following:

```
[req]
distinguished_name = req_distinguished_name
x509_extensions = myextensions

[req_distinguished_name]
countryName = Country Name (2 letter code)
countryName_default = GB
stateOrProvinceName = State or Province Name (full name)
stateOrProvinceName_default = Hants
localityName = Locality Name (eg, city)
localityName_default = Hursley
organizationName = Organization Name (eg, company)
organizationName_default = IBM United Kingdom
organizationalUnitName = Organizational Unit Name (eg, department)
organizationalUnitName_default = IBM MQ Development
commonName = Common Name (eg, Your Name)

[myextensions]
keyUsage = digitalSignature,nonRepudiation,dataEncipherment,keyEncipherment
extendedKeyUsage = emailProtection
```

2. AMS requires that both the certificate and private key are held in the same file. Issue the following command to achieve this:

```
cat $HOME/mycert.pem >> $HOME/private.pem
```

The `private.pem` file in $HOME now contains a matching private key and certificate, while the `mycert.pem` file contains all of the public certificates for which you can encrypt messages and validate signatures.

The two files need to be associated with your environment by creating a keystore configuration file, `keystore.conf`, in your default location.

By default, AMS looks for the keystore configuration in a `.mqs` subdirectory of your home directory.

3. In QShell create the `keystore.conf` file:

```
mkdir -p $HOME/.mqs
echo "pem.private = $HOME/private.pem" > $HOME/.mqs/keystore.conf
echo "pem.public = $HOME/mycert.pem" >> $HOME/.mqs/keystore.conf
echo "pem.password = unused" >> $HOME/.mqs/keystore.conf
```

IBM i *Creating a policy for AMS on IBM i*

Before creating a policy, you need to create a queue to hold protected messages.

Procedure

1. At a command line prompt enter;

```
CRTMQMQ QNAME(PROTECTED) QTYPE(*LCL) MQMNAME (mqmname)
```

where mqmname is the name of your queue manager.

Use the DSPMQM command to check that the queue manager is capable of using security policies. Ensure that **Security Policy Capability** shows **YES*.

The simplest policy you can define is an integrity policy, which is achieved by creating a policy with a digital signature algorithm but no encryption algorithm.

Messages are signed but not encrypted. If messages are to be encrypted, you must specify an encryption algorithm, and one or more intended message recipients.

A certificate in the public keystore for an intended message recipient is identified through a distinguished name.

2. Display the distinguished names of the certificates in the public keystore, `mycert.pem` in $HOME, by using the following command in QShell:

```
/QOpenSys/usr/bin/openssl x509 -in $HOME/mycert.pem -noout -subject -nameopt RFC2253
```

You need to enter the distinguished name as an intended recipient, and the policy name must match the queue name to be protected.

3. At a CL command prompt enter, for example:

```
SETMQMSPL POLICY(PROTECTED) MQMNAME (mqmname)SIGNALG(*SHA256) ENCALG(*AES256) RECIP('CN=..,
O=.., C=..')
```

where mqmname is the name of your queue manager.

Once the policy is created, any messages that are put, browsed, or destructively removed through that queue name are subject to the AMS policy.

Related reference

Display Message Queue Manager (DSPMQM)

Set MQM Security Policy (SETMQMSPL)

IBM i *Testing a policy for AMS on IBM i*

Use the sample applications provided with the product to test your security policies.

About this task

You can use the sample applications provided with IBM MQ , such as AMQSPUT4, AMQSGET4, AMQSGBR4, and tools such as WRKMQMMSG to put, browse, and get messages using the PROTECTED queue name.

Provided everything has been configured correctly, there should be no difference in application behavior to that of an unprotected queue for this user.

A user not set up for Advanced Message Security, or a user that does not have the required private key to decrypt the message will, however, not be able to view the message. The user receives a completion code of RCFAIL, equivalent to MQCC_FAILED (2) and reason code of RC2063 (MQRC_SECURITY_ERROR).

To see that AMS protection is in effect, put some test messages to the PROTECTED queue, for example using AMQSPUT0. You can then create an alias queue to browse the raw protected data while at rest.

Procedure

To grant necessary permissions to a user, run:

```
CRTMQMQ QNAME(ALIAS) QTYPE(*ALS) TGTQNAME(PROTECTED) MQMNAME(yourqm)
```

Browsing using the ALIAS queue name, for example using AMQSBCG4 or WRKMQMMSG, should reveal larger `scrambled` messages where a browse of the PROTECTED queue shows cleartext messages.

The scrambled messages are visible, but the original cleartext is not decipherable using the ALIAS queue, as there is no policy for AMS to enforce matching this name. Hence, the raw protected data is returned.

Related reference
Set MQM Security Policy (SETMQMSPL)
Work with MQ Messages (WRKMQMMSG)

Command and configuration events for AMS

With Advanced Message Security, you can generate command and configuration event messages, which can be logged and serve as a record of policy changes for auditing.

Command and configuration events generated by IBM MQ are messages of the PCF format sent to dedicated queues on the queue manager where the event occurs.

Configuration events messages are sent to the SYSTEM.ADMIN.CONFIG.EVENT queue.

Command events messages are sent to the SYSTEM.ADMIN.COMMAND.EVENT queue.

Events are generated regardless of tools you are using to manage Advanced Message Security security policies.

In Advanced Message Security, there are four types of events generated by different actions on security policies:

- "Creating security policies in AMS" on page 659, which generate two IBM MQ event messages:
  - A configuration event
  - A command event
- "Changing security policies in AMS" on page 660, which generates three IBM MQ event messages:
  - A configuration event that contains old security policy values
  - A configuration event that contains new security policy values
  - A command event
- "Displaying and dumping security policies in AMS" on page 661, which generates one IBM MQ event message:
  - A command event
- "Removing security policies in AMS" on page 662, which generates two IBM MQ event messages:
  - A configuration event
  - A command event

Enabling and disabling event logging for AMS

You control command and configuration events by using the queue manager attributes **`CONFIGEV`** and **`CMDEV`**. To enable these events, set the appropriate queue manager attribute to `ENABLED`. To disable these events, set the appropriate queue manager attribute to `DISABLED`.

Procedure

Configuration events

To enable configuration events, set **CONFIGEV** to ENABLED. To disable configuration events, set **CONFIGEV** to DISABLED. For example, you can enable configuration events by using the following MQSC command:

```
ALTER QMGR CONFIGEV (ENABLED)
```

Command events

To enable command events, set **CMDEV** to ENABLED. To enable command events for commands except **DISPLAY MQSC** commands and Inquire PCF commands, set the **CMDEV** to NODISPLAY. To disable command events, set **CMDEV** to DISABLED. For example, you can enable command events by using the following MQSC command:

```
ALTER QMGR CMDEV (ENABLED)
```

Related tasks

Controlling configuration, command, and logger events in IBM MQ

Command event message format for AMS

Command event message consists of MQCFH structure and PCF parameters following it.

Here are selected MQCFH values:

```
Type = MQCFT_EVENT;
Command = MQCMD_COMMAND_EVENT;
MsgSeqNumber = 1;
Control = MQCFC_LAST;
ParameterCount = 2;
CompCode = MQCC_WARNING;
Reason = MQRC_COMMAND_PCF;
```

Note: ParameterCount value is two because there are always two parameters of MQCFGR type (group). Each group consists of appropriate parameters. The event data consists of two groups, CommandContext and CommandData.

CommandContext contains:

EventUserID

Description:	The user ID that issued the command or call that generated the event. (This is the same user ID that is used to check the authority to issue the command or call; for commands received from a queue, this is also the user identifier (UserIdentifier) from the MD of the command message).
Identifier:	MQCACF_EVENT_USER_ID.
Data type:	MQCFST.
Maximum length:	MQ_USER_ID_LENGTH.
Returned:	Always.

EventOrigin

Description:	The origin of the action causing the event.
Identifier:	MQIACF_EVENT_ORIGIN.
Data type:	MQCFIN.

Values:	**MQEVO_CONSOLE** Console command - command line. **MQEVO_MSG** Command message from IBM MQ Explorer plugin.
Returned:	Always.

EventQMgr

Description:	The queue manager where the command or call was entered. (The queue manager where the command is executed and that generates the event is in the MD of the event message).
Identifier:	MQCACF_EVENT_Q_MGR.
Data type:	MQCFST.
Maximum length:	MQ_Q_MGR_NAME_LENGTH.
Returned:	Always.

EventAccountingToken

Description:	For commands received as a message (MQEVO_MSG), the accounting token (AccountingToken) from the MD of the command message.
Identifier:	MQBACF_EVENT_ACCOUNTING_TOKEN.
Data type:	MQCFBS.
Maximum length:	MQ_ACCOUNTING_TOKEN_LENGTH.
Returned:	Only if EventOrigin is MQEVO_MSG.

EventIdentityData

Description:	For commands received as a message (MQEVO_MSG), application identity data (ApplIdentityData) from the MD of the command message.
Identifier:	MQCACF_EVENT_APPL_IDENTITY.
Data type:	MQCFST.
Maximum length:	MQ_APPL_IDENTITY_DATA_LENGTH.
Returned:	Only if EventOrigin is MQEVO_MSG.

EventApplType

Description:	For commands received as a message (MQEVO_MSG), the type of application (PutApplType) from the MD of the command message.
Identifier:	MQIACF_EVENT_APPL_TYPE.
Data type:	MQCFIN.
Returned:	Only if EventOrigin is MQEVO_MSG.

EventApplName

Description:	For commands received as a message (MQEVO_MSG), the name of the application (PutApplName) from the MD of the command message.
Identifier:	MQCACF_EVENT_APPL_NAME.
Data type:	MQCFST.

Maximum length:	MQ_APPL_NAME_LENGTH.
Returned:	Only if EventOrigin is MQEVO_MSG.

EventApplOrigin

Description:	For commands received as a message (MQEVO_MSG), the application origin data (ApplOriginData) from the MD of the command message.
Identifier:	MQCACF_EVENT_APPL_ORIGIN.
Data type:	MQCFST.
Maximum length:	MQ_APPL_ORIGIN_DATA_LENGTH.
Returned:	Only if EventOrigin is MQEVO_MSG.

Command

Description:	The command code.
Identifier:	MQIACF_COMMAND.
Data type:	MQCFIN.
Values:	**MQCMD_INQUIRE_PROT_POLICY numeric value 205** **MQCMD_CREATE_PROT_POLICY numeric value 206** **MQCMD_DELETE_PROT_POLICY numeric value 207** **MQCMD_CHANGE_PROT_POLICY numeric value 208** These are defined in IBM MQ 8.0 `cmqcfc.h`
Returned:	Always.

CommandData contains PCF elements that comprised the PCF command.

Configuration event message format for AMS

Configuration events are PCF messages of standard Advanced Message Security format.

Possible values for the MQMD message descriptor can be found in Event message MQMD (message descriptor).

Here are selected MQMD values:

```
Format = MQFMT_EVENT
Peristence = MQPER_PERSISTENCE_AS_Q_DEF
PutApplType = MQAT_QMGR           //for both CLI and command server
```

Message buffer consist of MQCFH structure and the parameter structure that follows it. Possible MQCFH values can be found in Event message MQCFH (PCF header).

Here are selected MQCFH values:

```
Type = MQCFT_EVENT
Command = MQCMD_CONFIG_EVENT
MsgSeqNumber = 1 or 2          // 2 will be in case of Change Object event
Control = MQCFC_LAST or MQCFC_NOT_LAST     //MQCFC_NOT_LAST will be in case of 1 Change Object
event
ParameterCount = reflects number of PCF parameters following MQCFH
CompCode = MQCC_WARNING
Reason = one of {MQRC_CONFIG_CREATE_OBJECT, MQRC_CONFIG_CHANGE_OBJECT,
MQRC_CONFIG_DELETE_OBJECT}
```

The parameters following MQCFH are:

EventUserID

Description:	The user ID that issued the command or call that generated the event. (This is the same user ID that is used to check the authority to issue the command or call; for commands received from a queue, this is also the user identifier (UserIdentifier) from the MD of the command message).
Identifier:	**MQCACF_EVENT_USER_ID**
Data type:	MQCFST.
Maximum length:	MQ_USER_ID_LENGTH.
Returned:	Always.

SecurityId

Description:	Value of MQMD.AccountingToken in case of command server message or Windows SID for local command.
Identifier:	**MQBACF_EVENT_SECURITY_ID**
Data type:	MQCBS.
Maximum length:	MQ_SECURITY_ID_LENGTH.
Returned:	Always.

EventOrigin

Description:	The origin of the action causing the event.
Identifier:	**MQIACF_EVENT_ORIGIN**
Data type:	MQCFIN.
Values:	**MQEVO_CONSOLE** Console command - command line. **MQEVO_MSG** Command message from the IBM MQ Explorer plugin.
Returned:	Always.

EventQMgr

Description:	The queue manager where the command or call was entered. (The queue manager where the command is executed and that generates the event is in the MD of the event message).
Identifier:	**MQCACF_EVENT_Q_MGR**
Data type:	MQCFST
Maximum length:	MQ_Q_MGR_NAME_LENGTH
Returned:	Always.

ObjectType

Description:	Object type.
Identifier:	**MQIACF_OBJECT_TYPE**
Data type:	MQCFIN

Value:	**MQOT_PROT_POLICY** Advanced Message Security protection policy. **1019** - a numeric value defined in IBM MQ 8.0 or in the `cmqc.h` file.
Returned:	Always.

PolicyName

Description:	The Advanced Message Security policy name.
Identifier:	**MQCA_POLICY_NAME**.
Data type:	MQCFST.
Value:	**2112** - a numeric value defined in IBM MQ 8.0 or in the `cmqc.h` file.
Maximum length:	MQ_OBJECT_NAME_LENGTH.
Returned:	Always.

PolicyVersion

Description:	The Advanced Message Security policy version.
Identifier:	**MQIA_POLICY_VERSION**
Data type:	MQCFIN
Value	**238** - a numeric value defined in IBM MQ 8.0 or in the `cmqc.h` file.
Returned:	Always

TolerateFlag

Description:	The Advanced Message Security policy toleration flag.
Identifier:	**MQIA_TOLERATE_UNPROTECTED**
Data type:	MQCFIN
Value	**235** - a numeric value defined in IBM MQ 8.0 or in the `cmqc.h` file.
Returned:	Always.

SignatureAlgorithm

Description:	The Advanced Message Security policy signature algorithm.
Identifier:	**MQIA_SIGNATURE_ALGORITHM**
Data type:	MQCFIN
Value:	**236** - a numeric value defined in IBM MQ 8.0 or in the `cmqc.h` file.
Returned:	Whenever there is a signature algorithm defined in the Advanced Message Security policy

EncryptionAlgorithm

Description:	The Advanced Message Security policy encryption algorithm.
Identifier:	**MQIA_ENCRYPTION_ALGORITHM**
Data type:	MQCFIN
Value:	**237** - a numeric value defined in IBM MQ 8.0 or in the `cmqc.h` file.
Returned:	Whenever there is an encryption algorithm defined in the IBM MQ policy

SignerDNs

Description:	Subject DistinguishedName of allowed signers.
Identifier:	**MQCA_SIGNER_DN**
Data type:	MQCFSL
Value:	**2113** - a numeric value defined in IBM MQ 8.0 or in the `cmqc.h` file.
Maximum length:	Longest signer DN in the policy, but no longer then MQ_DISTINGUISHED_NAME_LENGTH
Returned:	Whenever defined in IBM MQ policy.

RecipientDNs

Description:	Subject DistinguishedName of allowed signers.
Identifier:	**MQCA_RECIPIENT_DN**
Data type:	MQCFSL
Value:	**2114** - a numeric value defined in IBM MQ 8.0 or in the `cmqc.h` file.
Maximum length:	Longest recipient DN in the policy, but no longer then MQ_DISTINGUISHED_NAME_LENGTH.
Returned:	Whenever defined in IBM MQ policy.

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Software Interoperability Coordinator, Department 49XA
3605 Highway 52 N
Rochester, MN 55901
U.S.A.

Programming interface information

Programming interface information, if provided, is intended to help you create application software for use with this program.

This book contains information on intended programming interfaces that allow the customer to write programs to obtain the services of IBM MQ.

However, this information may also contain diagnosis, modification, and tuning information. Diagnosis, modification and tuning information is provided to help you debug your application software.

Important: Do not use this diagnosis, modification, and tuning information as a programming interface because it is subject to change.

Trademarks

Part Number:

(1P) P/N: