Middleware Suite - Application Integration

NCI - Network Computing Interface

MQ Security Suite
Implementation Guide
Version: 3.1

T · · Systems ·

Impressum

Publisher	**T-Systems Enterprise Services GmbH** Computing Services & Solutions (CSS) System Products & Automation (MSY-PA)
Responsible	**T-Systems Enterprise Services GmbH** Computing Services & Solutions (CSS) System Products & Automation (MSY-PA) Fasanenweg 9, 70771 Leinfelden-Echterdingen, Germany cc.middleware@t-systems.com +49 89/1011-4687

Document information

Name of file

NCI MQSeries Security Suite

Version / Revision	Date	Revision
This edition relates to NCI version NCI 3.1	25/11/2005 08:49:00	220

- PNCI310/QZ05046 on z/OS
- PNCI310/REL1003 on Unix/Windows

List of available NCI documentation:

NCI Application Programming Reference
NCI Installation and Customization for Distributed Systems
NCI Installation and Customization for z/OS and OS/390
NCI MQ File Transfer Utilities
NCI MQ Security Suite
NCI SAP R/3 RFC Server Interface
NCI Additional Features

Additional License Information

Acknowledgment:

This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit. (http://www.openssl.org/)
This product includes cryptographic software written by Eric Young (eay@cryptsoft.com)
This product includes software written by Tim Hudson (tjh@cryptsoft.com)
This product includes software developed by the Apache Software Foundation (http://www.apache.org/)

Table of Contents

1 Introduction

Queue Managers need to ensure that they exchange messages with the correct partner. It is equally important for the sending queue manager to be sure of the receiver's identity as it is for the receiving queue manager to be sure of the sender's identity. This mutual authentication of queue managers has been implemented with MQ security exits as node-to-node security between sender and receiver channels.

In an MQ client-server environment, there is a client-connection channel which passes MQ API commands to a server-connection channel. The server-connection channel serves as a proxy to the client application and executes the client commands. The userid which the server MCA uses to access MQSeries can either be defined permanently for the server-connection or passed dynamically from the MQ client. In order to ensure, that only authorized users gain access to MQSeries resources an authorization check at the server site for the userid provided by the client has been implemented with the MQ security exits.

The NCI MQ Security Suite provides the following features

- Node-to-Node security between MQSeries servers
- Application-level security for MQSeries clients
- Encryption of transmitted messages
- Compression of transmitted messages

which will be described in the following chapters.

These features have been implemented via the following MQSeries exits:

Security exit	Node-to-node and application security
Send and receive exits	Encryption and compression
Channel auto-definition exit	Propagation of channel parameters in a cluster environment

1.1 Node-to-node Security between MQSeries Servers

In order to connect MQSeries servers, sender and receiver channels have to be configured. Addressing of a receiver channel by a sender channel occurs via IP address, port number and channel name. Thus, in an intranet, in principle each pair of MQSeries servers can connect to each other.

To prevent this, the NCI MQ security suite provides MQSeries security exits, that can be installed at both ends of the communication path to authenticate the sender channel by the receiver channel and vice versa. The authentication occurs at channel startup, when an encrypted key will be transmitted from the sender to the receiver, where it will be compared with the key specified in the exit configuration file of the receiver. The same process takes place in the opposite direction.

1.2 Application-level Security for MQSeries Clients

An MQSeries client is a component of MQSeries, that can be installed on its own, on a separate machine from the base product and server. MQSeries applications can run on an MQSeries client and can connect, by means of a communications protocol like TCP/IP, to one or more remote queue managers.

Communication between MQSeries clients and servers occurs via MQI channels. Channels have to be set up at each end of the communication link. The channel on the server site, the so-called **server-connection channel**, will be defined by using an MQSeries command on the server machine. To define the channel at the client site, the so-called **client-connection channel**, the following two methods can be used:

1. By setting the environment variable **MQSERVER**, a minimal channel can be defined. It specifies the location of the MQSeries server and the communication method to be used.

   Note: Using this method it is not possible to specify a security exit. Therefore method 2 must be used in order to implement security with the NCI MQ Security Suite for MQSeries clients.

2. By using MQSeries commands to define the **client-connection channel** at the server site while specifying a security exit in the channel definition. The resulting **client channel definition table** must be made accessible to the MQSeries client. How this can be done is platform-specific and described in detail in the IBM manual "MQSERIES Clients, Chapter 8. Using Channels". Specification of the channel table at the client site occurs via the environment variables **MQCHLLIB** and **MQCHLTAB**.

   Note: The **MQSERVER** environment variable takes precedence, when specifying client connection channels. In order to use the NCI MQ Security Suite the **MQSERVER** must not be set.

In both cases addressing of the **server-connection channel** occurs via IP address, port number and channel name. If this information is known in an intranet, a client application can access every MQSeries queue to which the **server-connection channel** has access.

To eliminate this security hole, the NCI MQ Security Suite must be installed at both ends of the communication link, that is the NCI security exits must be specified for the server-connection channel as well as the client-connection channel.

If the NCI MQ Security Suite has been installed and configured for a channel connection between an MQSeries client and server, security information, that is userid and password provided by the client application, will be transmitted in encrypted form to the server. The security exit at the server site performs the authentication verification according to the settings in the exit configuration file at the server site. Authentication verification can be done versus:

1. the security product of the operating system, that is RACF on z/OS or /etc/passwd on Unix systems
2. the **MCAUSER** defined for the **server-connection channel** and a password specified with the **ClientPassword** parameter in the exit configuration file.

An NCI client application program can specify the security information via the API calls **NCISetUserId** and **NCISetPwd** or the sideinformation keywords **UserId** and **Pwd** , whereas a native MQSeries client program must set the environment variables **MQ_USER_ID** and **MQ_PASSWORD**. If an error occurs during authentication verification, an NCI program can obtain detailed error information via the API calls

NCIGetErrorReasonCode NCIGetErrorMsgText, whereas a native MQSeries program can inspect the environment variable NCIREASONCODE. See the NCI Application Reference Manual for details to the calls and reason codes.

1.3 Encryption of transmitted messages

In order to prevent messages from being tapped when transmitted between two MQSeries nodes (client or server), the NCI MQ Security Suite can be installed at each end of the communication link as MQSeries send and receive exit respectively. The exit encodes and decodes message segments by using DES encryption (56 Bit).

1.4 Compression of transmitted messages

In order to reduce the size of transmitted messages the NCI MQ Security Suite can be installed at each end of the communication link as MQSeries send and receive exit respectively. The exit compresses and expands message segments by using one of several available compression methods.

2 Installation

The NCI MQ Security Suite is provided in the form of a dynamically loadable object (a DLL on Windows NT, a shared library on Unix systems and a load module on z/OS). The exits must be installed on **both** ends of the communication link, i.e. **sender** and **receiver channel** or respectively **client-connection** and **server-connection channel**. In the case of the MQSeries security exit, if only the exit at the receiving end is installed, connection requests from sender or client-connection channels are always rejected.

In order to install and configure the exits the MQSeries command **DEFINE CHANNEL** with the **REPLACE** option should be used.

In the case of node-to-node or application-level security the exit name must be specified with the **SCYEXIT** keyword for every channel involved in the communication.

For message encryption and compression there are two cases to consider. For communication between two MQSeries servers the exit name must be specified with the **SENDEXIT** keyword for the sender channel and the **RCVEXIT** keyword for the receiver channel. For communication between an MQSeries client and an MQSeries server the exit name must be specified for the **server-connection channel and the client-connection channel** with the **SENDEXIT** as well as the **RCVEXIT** keyword. The format of the exit name is as follows:

libncimcx.a(ChannelExit)	on AIX
libncimcx.sl(ChannelExit)	on HP/UX
libncimcx.so(ChannelExit)	on Sun Solaris
ncimcx(ChannelExit)	on Windows NT
ncimcx	on z/OS

2.1 Installation on Unix and Windows systems

For Unix and NT systems either the full path name has to be specified or alternatively, the path name can be specified in the **ExitPath** stanza of the **QM.INI** file for an MQSeries server or the **MQS.INI** file for an MQSeries client.

Note: The full path name, including library name and entry point, can have a maximum length of 32 characters.

Note: Starting with MQSeries 5.2 for Windows the QM.INI file has been replaced by registry entries, which can be changed by selecting *properties* for the queue manager in the MQ-Services console. The configuration of the exits will be specified in a file, whose filename and path must be specified with the **SCYDATA** keyword of the channel definition.

For example the definition of a server-connection channel on Sun Solaris may look as follows:

Example of a server-connection channel on Sun Solaris

```
define  channel (MQ01.CLIENT.CHL) chltype (svrconn) trptype (tcp) +
        mcauser('mqm') +
        scyexit('/home/mq/libncimcx.so(ChannelExit)') +
        scydata('/home/mq/MQ01EXIT.def') +
        sendexit('/home/mq/libncimcx.so(ChannelExit)') +
        rcvexit('/home/mq/libncimcx.so(ChannelExit)') +
        descr('Server-connection channnel for MQ clients') +
        replace
```

2.1.1 Special Consideration for Unix Systems

On unix systems with a shadow password file, currently Sun Solaris, AIX 4.3 and Linux, in order for the authentication check to work, the owner ID of the module **amqcrsta** in the **mqm/bin** directory has to be changed to **root** and the sticky bit has to be turned on.

If this cannot be done for any reason an authentication check can only be done versus the MCA userid specified in the server-connection channel definition and the password specified in the configuration file.

2.2 Installation on z/OS systems

On z/OS systems in order to allow authentication checking of MQSeries clients the program **CSQXJST** started by the channel initiator startup procedure **xxxxCHIN** has to be replaced by **NCISXIPC**, which installs a PC routine for authentication checking and subsequently calls the channel initiator program **CSQXJST**.

The following load modules have to be made available to the channel initiator startup procedure:

NCIMCX Dataset specified with CSQXLIB DD-card of xxxxCHIN procedure.

NCISXIPC LINKLIB or STEPLIB concatenation.

NCISXAUT LINKLIB or STEPLIB concatenation.

Note, that **NCISXIPC** and **NCISXAUT** must run APF-authorized.

Copy the required load modules from the SMP/E target library to the to the libraries mentioned above.

3 Configuration of the NCI MQ Security Suite

The configuration of the NCI MQ Security Suite is different for channels running at an MQ server site compared to those running at an MQ client site, i.e. the client-connection channel. In the following description of the channel configuration keywords and parameters it is indicated for which channel type, i.e. sender channel, receiver channel or server-connection channel the particular keyword is applicable.

3.1 **Configuration for channels running at the MQ server site**

The configuration of sender, receiver and server-connection channels is accomplished by specifying configuration parameters in an exit configuration file. The name and path of the exit configuration file must be specified in the SCYDATA field of the channel definition (see the example above). The syntax of the parameters is as follows:

Each keyword must begin and end on a separate line. All data between the opening and the closing bracket is treated as the keyword value. Leading or trailing blanks are not stripped!

Note: Keywords are not case sensitive. However the keyword values are all case sensitive. To prevent errors, code the keyword values in exactly the same syntax they are described in the reference chapters.

Lines starting with a '*' are treated as comment lines.

Each exit configuration file must contain at least one named symbolic section (keyword ChannelName). The NCI MQ Security Suite does not support a default section. An exit configuration file may contain multiple sections, offering the flexibility to define different channel definitions within the same exit configuration file.

All keywords defined in the exit configuration file before the first channel section are treated as global keywords. Global keywords are propagated to all subsequent channel sections.

If a global keyword is also defined within a channel section, the global definition will be overridden by the definition in the channel section.

The following keywords can be specified.

Channel Name


Required keyword.

Defines the beginning of a group of parameters applying to the specified channel. The end of parameters belonging to this channel is either marked by the next *ChannelName* keyword or by the end of the file.

```
Client Authentication

>>--ClientAuth(--+-YES-+--)--><
                 !     !
                 +-NO--+
```

Optional keyword. Applicable only for server-connection channels.

Specifies for application-level security if security information, i.e. userid and password, must be provided by the client application.

Option	Description
YES	Security information must be provided
NO	Security information is optional.

Note: Note that, no matter what value has been specified for the *ClientAuth* keyword, if userid and/or password has been provided by the client application an authorization check will be performed by the security exit at the server site. This allows to use one server-connection channel for clients that access security-critical information at the server site as well as for clients that access uncritical information. The authorization checking towards queues will always be done by operating system facilities like *RACF* on *Z/OS* or *OAM* on distributed platforms.

```
Client Authentication Method

>>--ClientAuthMethod(--+-OS-------+--)--><
                       !          !
                       +-EXITDATA-+
```

Optional keyword. Required if *YES* has been specified for the *ClientAuth* keyword. Applicable only for server-connection channels.

Specifies how, in the case of application-level security, the authentication checking of an MQ client will be performed.

Option	Description
OS	Userid and password provided by the MQ client application will be authenticated by the security system at the MQ server site.
EXITDATA	Authentication will be performed using the value specified for the *MCAUSER* keyword of the channel definition of the server-connection channel and the password specified for the *ClientPassword* keyword of the channel exit configuration file. This allows to avoid having to define every client userid to the security system at the server site.

```
Client Password

>>--ClientPassword(--string--)--><
```

Optional keyword. Required if *YES* has been specified for the *ClientAuth* keyword and *EXITDATA* has been specified for the *ClientAuthMethod* keyword. Applicable only for server-connection channels.

Specifies the password that will be used for authentication checking of an MQ client when *ClientAuthMethod* has been set to *EXITDATA*.

```
Compatibility Mode

>>--CompatMode(--+-YES-+--)--><
                 !     !
                 +-NO--+
```

Optional keyword.

Specifies if compatibility with version 1.0 of the the security exit at the client site is required.

Note: Note that for communication with a version 1.0 security exit a built-in key will be used for encryption instead of that specified with the *EncyptionKey:* keyword.

```
Compression Type

>>--CompressionType(--+-NONE-----+--)--><
                      !          !
                      !-HUFF-----!
                      !          !
                      !-HUFF-RLE-!
                      !          !
                      !-RLE------!
                      !          !
                      +-LZ-------+
```

Optional keyword.

Specifies the compression method used to compress transmitted messages.

Option	Description
NONE	No compression.
HUFF	Compression using huffman algorithm
HUFF-RLE	Compression using run-length encoding before the huffman algorithm is applied
RLE	Compression using run-length encoding
LZ	Compression using the Lempel-Ziv algorithm

```
Effective User

>>--EffectiveUser(--+-PROPAGATE-+--)--><
                    !           !
                    +-SURROGATE-+
```

Optional keyword.

Specifies which userid will be used for authorization checking at the server site for a client request.

Option	Description
PROPAGATE	The userid provided by the MQ client via security information will be used.
SURROGATE	The userid specified with the MCAUSER keyword of the server-connection

	channel definition will be used.

```
Encryption Type

>>--EncryptionType(--+-NONE-+--)--><
                     !      !
                     +-DES--+
```

Optional keyword.

Specifies the encryption method used to encrypt transmitted messages.

Option	Description
NONE	No encryption
DES	Encryption according to the DES algorithm

```
Encryption Key

>>--EncryptionKey(--string--)--><
```

Optional keyword.

If encryption of transmitted messages has been requested via keyword *EncryptionType*, *EncryptionKey* specifies the DES key to be used for encryption and decryption respectively. The key can be specified in hexadecimal, octal, binary or decimal format and must have a length of 8 bytes.

For node-to-node security *EncryptionKey* specifies the key to be used for mutual authentication.

Example:

hexadecimal	EncryptionKey(0x6465666768693435)
octal	EncryptionKey(0o621453146355032232065)
decimal	EncryptionKey(0d7234300970759959605)
decimal	EncryptionKey(0b100110102220111100111100...)

```
Library Name

>>--LibraryName(--OS independent part of library name--)--><
```

Optional keyword. Required if **DYNAMIC** has been specified for **ScyExit**, **SendExit** or **ReceiveExit** keyword. Applicable only for auto-definition channels.

Specifies the operating system-independent part of the library name of the security, send and receive exit, i.e. **ncimcx**.

```
Library Entry

>>--LibraryEntry(--OS entry-point name for exit libary--)--><
```

Optional keyword. Required if **DYNAMIC** has been specified for **ScyExit**, **SendExit** or **ReceiveExit** keyword. Applicable only for auto-definition channels.

Specifies the entry-point name for the security, send and receive exit, i.e. **ChannelExit**.

```
Receive Data

>>--RcvData(--+-path and name of config file-+--)--><
```

Optional keyword. Applicable only for auto-definition exit.

Specifies the path and name of the configuration file for the receive exit when the auto-definition exit will be used in an MQ cluster environment and the **DYNAMIC** keyword has been specified for the **RcvExit** parameter.

```
Receive Exit

>>--ReceiveExit(--+-[path]libname(entrypoint)-+--)--><
                  !                           !
                  +-DYNAMIC-------------------+
```

Optional keyword. Applicable only for auto-definition exit.

Specifies the path, library name and entry point of the receive exit for use by the auto-efinition exit. When specifying the **DYNAMIC** keyword, the auto-definition exit will generate the library name according to the platform from the values specified with the **LibraryName** and **LibraryEntry** keywords. In this case the library has to reside in the standard path */var/mqm/exits* for unix systems or \MQSeries\Exits for windows respectively.

```
Security Data

>>--ScyData(--+-path and name of config file-+--)--><
```

Optional keyword. Applicable only for auto-definition exit.

Specifies the path and name of the configuration file for the security exit when the auto-definition exit will be used in a MQ cluster environment and the **DYNAMIC** keyword has been specified with the **ScyExit** parameter.

```
Security Exit

>>--ScyExit(--+-[path]libname(entrypoint)-+--)--><
              !                         !
              +-DYNAMIC-------------------+
```

Optional keyword. Applicable only for auto-definition exit.

Specifies the path, library name and entry point of the security exit for use by the auto-definition exit. When specifying the **DYNAMIC** keyword, the auto-definition exit will generate the library name according to the platform from the values specified with the **LibraryName** and **LibraryEntry** keywords. In this case the library has to reside in the standard path */var/mqm/exits* for unix systems or \MQSeries\Exits for windows respectively.

```
Send Data

>>--SendData(--+-path and name of config file-+--)--><
```

Optional keyword. Applicable only for auto-definition exit.

Specifies the path and name of the configuration file for the send exit when the auto-definition exit will be used in a MQ cluster environment and the **DYNAMIC** keyword has been specified with the **SendExit** parameter.

```
Send Exit

>>--SendExit(--+-[path]libname(entrypoint)-+--)--><
               !                          !
               +-DYNAMIC-------------------+
```

Optional keyword. Applicable only for auto-definition exit.

Specifies the path, library name and entry point of the send exit for use by the auto-definition exit. When specifying the **DYNAMIC** keyword, the auto-definition exit will generate the library name according to the platform from the values specified with the **LibraryName** and **LibraryEntry** keywords. In this case the library has to reside in the standard path */var/mqm/exits* for unix systems or \MQSeries\Exits for windows respectively.

```
Trace Mode

>>--TraceMode(--+-NONE--+--)--><
                !       !
                !-ERROR-!
                !       !
                !-INFO--!
                !       !
                +-ALL---+
```

Optional keyword. Applicable only for server-connection channels.

Specifies, if and what messages will be written to the trace file.

Option	Description
NONE	Logging and tracing turned off.
ERROR	Error messages will be written.
INFO	Error and info messages will be written. Recommended setting.
INFO	Error, info and trace messages will be written.

```
Trace File

>>--TraceFile(--path and name of trace file--)--><
```

Optional keyword. Required if log or trace messages have been requested via the *TraceMode* keyword. Applicable only for server-connection channels.

3.1.1 Errors at the sending site

If an error occurs on the sending end a non-persistent message gets lost, by default, since the message has been got outside syncpoint. Closing the channel will not cause the message to be rolled back to the transmission queue. Too ensure, that messages are persistent when using the NCI API, specify DEFPSIST(YES) when defining the remote queue on the sending end.

3.2 Configuration of the Client-connection Channel

The configuration of a client-connection channel will be done via the channel definition at the server site. The resulting channel table must be made available to the MQSeries client. See the IBM book *MQSeries Clients* on how to accomplish this task. The parameters will be specified via the SCYDATA-keyword according to the following syntax:

```
DEFINE CHANNEL ... SCYDATA

>>--E(--+-0-+--),C(--+-0-+--),K(--+-DES encryption key, hexadecimal coded-+-->
        !   !         !   !
        +-1-+         !-1-!
                      !   !
                      !-2-!
                      !   !
                      !-3-!
                      !   !
                      +-4-+

>--)--><
```

The usage of the keywords is as follows:

E Specifies the encryption method.

- 0 No encryption
- 1 Encryption according to the DES algorithm.

C Specifies the compression method

- 0 No compression

1 Compression using run-length encoding

2 Compression using the huffman algorithm

3 Compression using the Lempel-Ziv algorithm

4 Compression using the run-length encoding before huffman algorithm is applied

K Specifies the DES encryption key, hexadecimal coded

For example the definition of a client-connection channel on Windows could look as follows:

Example of a client-connection channel for Windows

```
define  channel (MQ01.CLIENT.CHL) chltype (clntconn) trptype (tcp) +
        conname('1.2.3.4(1415)') qmname('MQ01') +
        scyexit('c:\ncimcx(ChannelExit)') +
        scydata('E(1),C(3),K(6465666768693435)') +
        sendexit('c:\ncimcx(ChannelExit)') +
        rcvexit('c:\ncimcx(ChannelExit)') +
        descr('Connection for MQ clients') +
        replace
```

Note: Note, due to the limited size of the **scydata** parameter, only hexadecimal notation (without the preceding *0x*) is allowed for the encryption key of the client-connection channel.

3.2.1 Interface to the security exit at the client site

The MQ client application has to provide the security information, i.e. userid and password, if the security exit at the server-connection channel has been configured to perform an authentication check for every client request (the value for the *ClientAuth* has been specified as *YES*). The security information will be transmitted in encrypted format from the security exit at the client site to the security exit at the server site where the authentication check will be performed.

Also tracing can be requested by the client application and a reasoncode indicating the outcome of the authentication check can be retrieved.

How to specify userid and password, how to request tracing and how to retrieve the reasoncode for NCI and non-NCI applications is explained under the following headings.

3.2.1.1 Interface for NCI Applications

The security information can be specified by either by the NCI API calls **NCISetUserid** and or the corresponding NCI sideinformation parameters **UserId** and **Pwd**.

Tracing can be specified by using the NCI API calls **NCISetTraceFile** and **NCISetTraceOpt** or the corresponding NCI sideinformation parameters **TraceFile** and **TraceOpt**.

The reasoncode indicating the outcome of the authentication check can be retrieved with the NCI API call **NCIGetErrorReasonCode**.

See the *NCI Application Programming Reference* for details.

3.2.1.2 Interface for non-NCI Applications

The security information can be specified by setting the environment variables **MQ_USER_ID** and **MQ_PASSWORD** either by a script or by the MQI program.

Tracing will can be activated by setting the environment variable **NCITRACEOPT** to the value **SECEXIT** and the environment variable **NCITRACEFILE** to the path of a trace file.

The reasoncode indicating the outcome of the authentication check by the security exit at the server site can be retrieved from the environment variable **NCIREASONCODE**.

3.2.1.3 Reasoncodes from the authentication check

Reason	Description
112	NCI_RCC_SEC_USERID_REQUIRED Userid is required to allow access.
116	NCI_RCC_SEC_PWD_REQUIRED Userid and password are required to allow access.
120	NCI_RCC_SEC_USERID_NOT_DEFINED The specified userid is not defined on server side.
128	NCI_RCC_SEC_USERID_REVOKED The specified userid has been revoked on server side.
132	NCI_RCC_SEC_PWD_EXPIRED The specified password has expired.
136	NCI_RCC_SEC_PWD_INVALID The specified password is invalid.
144	NCI_RCC_SEC_UNKNOWN_DENY An unexpected error occured on server site. Request will be denied.

4 Configuration Verification

4.1 Node-to-node security

1. Define a sender channel and a receiver channel for different queue managers by using MQSeries commands. Define the exit configuration file according to the provided sample. Specify a tracefile, set TraceMode(ALL), specify an the same encryption key for both channels in the exit configuration file. Specify the path to the security exit loadable object in the SCYEXIT field of the channel definition according to the rules of the particular operating system:

```
Example for defining a sender and receiver channels on z/OS

define  channel (MQ01.MQ02) chltype (sdr) trptype (tcp) +
        conname (11.222.333.44) xmitq(MQ01.MQ02.XMITQ) +
        scyexit('NCIMCX') +
        scydata('HLQ.MQ01.INPUT(EXITCONF)') +
        descr('Sender Channel to MQ02') +
        replace

define  channel (MQ02.MQ01) chltype (rcvr) trptype (tcp) +
        scyexit('NCIMCX') +
        scydata('HLQ.MQ01.INPUT(EXITCONF)') +
        descr('Receiver Channel from MQ01') +
        replace
```

2. Define the configuration file *HLQ.MQ01.INPUT(EXITCONF)* according to the sample provided in the appendix.
3. Issue the following command to verify the configuration:

```
Command to verify the security exit configuration

START CHANNEL(MQ01.MQ02)
```

4. Change the password of the receiver channel to verify that the security exit works as expected.

4.2 Application-level security

1. Define the server-connection channel by using MQSeries commands at the server site. Define the MCAUSER field as USER1. Define the exit configuration file according to the provided sample. Specify ClientAuth(YES), ClientAuthMethod(OS), EffectiveUser(PROPAGATE), TraceMode(ALL) and a trace file in the exit configuration file. Specify the path to the security exit loadable object in the SCYEXIT field according to the rules of the server operating system

Example for defining a server-connection channel on z/OS

```
define  channel (MQ01.CLIENT.CHL) chltype (svrconn) trptype (tcp) +
        mcauser('USER1') +
        scyexit('NCIMCX') +
        scydata('HLQ.MQ01.INPUT(EXITCONF)') +
        descr('Connection for MQSeries clients') +
        replace
```

2. Define the client-connection channel by using MQSeries commands also at the server site. Specify the path to the security exit loadable object in the SCYEXIT field according to the rules of the client operating system.

Example for defining a client-connection channel for Windows NT on z/OS

```
define  channel (MQ01.CLIENT.CHL) chltype (clntconn) trptype (tcp) +
        conname (11.222.333.44) qmname('MQ01') +
        scyexit('c:\Programme\MQSeries\exits\ncimcx(ChannelExit)') +
        scydata('E(1),C(3),K(6465666768693435)') +
        replace
```

3. Note: Note that the backslash character has eventually to be replaced, depending on the codepage used for your 3270 emulation software on z/OS, for example for codepage 273 (Germany) the backslash character has to be replaced by Ö.
4. Define the exit configuration file *HLQ.MQ01.INPUT(EXITCONF)* at the server site according to the provided sample.
5. Transfer the resulting channel definition table in binary form to the client site. See the IBM manual "MQSeries Clients" on how this must be done for the particular platform.
6. Specify name and path of the channel definition table by using the environment variables MQCHLLIB and MQCHLTAB.

Example for specifying environment variables on Windows NT

```
set MQChllib=C:\
set MQChltab=AMQCLCHL.TAB
```

7. Issue the following commands to verify the configuration:

Commands to verify the security exit configuration

```
. ./setenv.sh     (only once)
NCIPING -a MQ -1 <qmgr> -2 <queue> -yN -u USER1 -p PWD1
```

8. A message will be placed in the specified queue, if USER1 is authorized for it. Issue other NCIPING-commands where the userid and/or password is omitted or wrong to verify that the security exit works as expected.

4.3 Encryption and Compression

1. Define a sender channel and a receiver channel for different queue managers by using MQSeries commands.

Example for defining a sender and receiver channels on z/OS

```
define  channel (MQ01.MQ02) chltype (sdr) trptype (tcp) +
        conname (11.222.333.44) xmitq(MQ01.MQ02.XMITQ) +
        sendexit('NCIMCX') +
        senddata('HLQ.MQ01.INPUT(EXITCONF)') +
        descr('Sender Channel to MQ02') +
        replace

define  channel (MQ02.MQ01) chltype (rcvr) trptype (tcp) +
        rcvexit('NCIMCX') +
        rcvdata('HLQ.MQ02.INPUT(EXITCONF)') +
        descr('Receiver Channel from MQ01') +
        replace
```

2. Define a configuration file according to the provided sample with encryption and compression activated.
3. Issue the following commands to verify the configuration:

Commands to verify the security exit configuration

```
On the sender side to put a message on a queue:
NCIPING -a MQ -1 <qmgr> -2 <remote queue> -yN -cY
On the receiver side to retrieve a message from a queue:
NCIPONG -a MQ -1 <qmgr> -2 <local queue> -w5
```

4. Inspect the log files on either side.

5 Node-to-node security in an MQ cluster environment

In an MQ cluster environment an auto-definition exit must be used in order to change parameters, such as the security exit library name, dynamically. The NCI MQ Security Suite provides an auto-definition exit.

In a cluster environment channel definitions are distributed via the cluster repository. A sending queue manager dynamically defines his cluster sender channel by using the channel definition of his receiver channel stored in the cluster repository. The library names of any exits defined in channel definitions are platform-specific and thus must be changed by the auto-definition exit.

In order to work correctly, the channels, which have to be changed by the auto-definition exit must have send and receive exits specified in the channel definition of their receiver channel. Otherwise the auto-definition exit is not able to put any value into these fields dynamically. This implies, that its not possible to use the security exit in a heterogenous cluster environment without also specifying the send and receive exits.

The auto-definition exit can be defined by using the following *runmqsc*-command:

Solaris: alter qmgr chadexit('libncimcx.so(AutoDefExit)')

AIX: alter qmgr chadexit('libncimcx.a(AutoDefExit)')

HP-UX: alter qmgr chadexit('libncimcx.sl(AutoDefExit)')

Windows: alter qmgr chadexit('ncimcx.(AutoDefExit)')

The auto-definition exit will be configured by an exit configuration file. This could be the same as those for the security exit, but it also can be separate.

There are two possibilities to pass the name of the exit configuration file to the auto-definition exit:

1. Default name and path

   Unix: /var/mqm/config/ncimcx (recommended when using *inetd* as listener)

   Windows: ...\MQSeries\Config\ncimcx

2. Environment variable **NCI_MCXCONF**=[path]name

There are a number of additional keywords for configuration of the auto-definition exit (*ScyExit, SendExit, RcvExit, LibraryName, LibraryEntry*). See the section, where the configuration keywords are described for details.

6 Application-level Securit & Java

According to the MQSeries client implementation for other languages IBM also provides a MQSeries client implementation based on 100% pure Java. For the Java client implementation NCI delivers almost the same security features as we do for the non Java world.
The NCI MQSeries Java Security implementation is not restricted to NCI users (application code that uses NCI Java classes). It could also be used within non NCI Java code.
Anyway we recommend to use the NCI Java API because it makes the programming and handling of application programs more easy.

Note: The IBM MQSeries Java implementation also provides the functionality to connect to a local queue manager. However the security features are only supported when connecting to a remote queue manager.

6.1 MQSeries Java implementation - Differences to other languages

There are some differences between IBMs MQSeries Java implementation and the implementation for other languages/platforms. Some of those affect the handling of the security and are therefore mentioned here.

The environment variables MQCHLTAB and MQCHLLIB, used in non Java environments to set the client connection properties, are NOT supported. Therefore there is no need to generate a client channel table and to distribute this table. Instead the client connection properties have to be set directly in the Java code.
If you use the NCI Java classes you do not have to worry about this. Enabling NCI MQSeries security is just to set one more option (see "Enabling NCI MQSeries security using NCI Java classes" below).

6.2 Enabling NCI MQSeries security using NCI Java classes

Enabling the security when programming with the NCI Java classes is very simple. Most of the security stuff is handeled internally. The security and other functions like message encryption and message compression is enabled by setter methods of the used Nci object instance. When working with NCI sideinfo files it is also possible to use the according sideinfo keywords.
In the case of an error that was caused by a security failure the usual NCI error handling can be done. That means, errors are passed back to the caller via the different NCI exceptions. The error reason and message text can be retrieved with the getErrorReasonCode() and getErrorMessageText() methods. For example, if the connection to the queue manager fails because a wrong password was supplied a NciInfoException is thrown and the error reason is set to RCC_SEC_PWD_INVALID (136).

Step by step introduction.

- Enable the NCI MQSeries security with the keyword "YES" as the last parameter on the setMQServer(...,...,...,"YES") method call. Or use the sideinfo keyword MQServer(...,...,...,YES)
- Optional - if desired, enable message compression with the method setMQExitDataCompress(...), or use the sideinfo keyword MQExitDataCompress(...).
  The method parameter specifies the type of algorithm used for the compression. For a list of algorithms currently supported please refer to the NCI Javadoc documentation.
- Optional - if desired, enable message encryption with the method setMQExitDataEncrypt("YES"), or use the sideinfo keyword MQExitDataCompress(YES). If enabled messages are encrypted with a 56 bit fixed DES key.

- Add the Jar archives ncimqexits.jar and joedm.jar to your application classpath.
- Configure the channel exit on the server side.
  Define all 3 types of exits for the channel.

  Note: It is important to define all 3 exit types (security, receive and send exit) in order to allow Java clients to connect to the server channel in secure mode. This is necessary, even if you neither use message compression nor message encryption. This is because of the internal information flow between the Java exit code and server channel exit code. We intend to change this behaviour in the future.

- Configure the channel exit definition file.
  The channel exit definition file is specified in the SCYDATA field of the channel definition.
  The following settings for the channel are important:
  - CompatMode(YES) - keyword is necessary to accept Java clients.

  All other settings like compression and encryption depend on your needs. Please refer to "Configuration for channels running at the MQ server site" on page 11

6.2.1 Sample - Using NCI

This sample illustrates how to enable the NCI MQSeries security using the NCI Java classes.

Sample Java code

```
import com.debis.nci.*;

public class NciSampMQ {

{
public NciSampMQ() {
 super();
}

public static void main(java.lang.String[] args) {

NciMessage outMessage  = new NciMessage("Message 1");
Nci nci                = new Nci(); // Create an instance of NCI

try
{
  nci.setAddrType("MQ");                // use protocol MQSeries
  nci.setPrimAddrInfo("MQ01");          // name of queue manager
  nci.setSecAddrInfo("APP.TEST.QUEUE"); // name of queue
                                        // enable client connection
  nci.setMQServer("MQ01.CLIENT.CHL","53.113.127.88","1414","YES");
  nci.setUserId("bond");                // provide userid
  nci.setPwd("007");                    // provide password
  nci.setMQExitDataCompress("HUFF");    // enable message compression

  nci.put(outMessage);                  // put the message

  System.out.println("put successful");
}
catch (NciInfoException e)
{
   System.out.println("put failed. Reason: " + nci.getErrorMsgText());
}
catch (Exception e)
{
  e.printStackTrace();
}

}//end of main
}//end of class
```

Sample channel definition (Platform z/OS)

```
define  channel (MQ01.CLIENT.CHL) chltype (svrconn) trptype (tcp) +
        mcauser('USER1') +
        scyexit('NCIMCX') +
        scydata('HLQ.MQ01.INPUT(EXITCONF)') +
        sendexit('NCIMCX') +
        rcvexit('NCIMCX') +
        descr('Secure connection for MQSeries Java clients') +
        replace
```

Sample Exit Definition File

```
************************************
* Configuration for NCI MQSeries     *
* Channel Exits.                     *
************************************

...

ChannelName(MQ01.CLIENT.CHL)
  ClientAuth(YES)
  ClientAuthMethod(OS)
  EffectiveUser(PROPAGATE)
  EncryptionType(NONE)
  CompressionType(NONE)
  CompatMode(YES)
  TraceMode(ALL)

...
```

6.3 Enabling NCI MQSeries security in non NCI environments

As already mentioned above. The usage of the NCI MQseries Java security is not restricted to NCI users. The NCI MQSeries Java security implementation may also be used in non NCI programs. After reading the following set-by-setp introduction and studying the subsequent sample code it should be easy to enable it.

Step by step introduction.

- Instanciate a Nci MQSeries Security object (class NciMQSecurityExit).
- Use the setter methods to pass the security properties like userid, password, compression, ... to this instance.
  Please refer to the NCI Javadoc documentation to get more info about the class NciMQSecurityExit.
- Enable the MQSeries security by setting the appropriate MQSeries connection properties (see sample below).
- Add the Jar archives nci.jar, ncimqexits.jar and joedm.jar to your application classpath.
- Configure the channel exit on the server side.
  Define all 3 types of exits for the channel.

  Note: It is important to define all 3 exit types (security, receive and send exit) in order to allow Java clients to connect to the server channel in secure mode. This is necessary, even if you neither use message compression nor message encryption. This is because of the internal

information flow between the Java exit code and server channel exit code. We intend to change this behaviour in the future.

- Configure the channel exit definition file.
  The channel exit definition file is specified in the SCYDATA field of the channel definition.
  The following settings for the channel are important:
  - CompatMode(YES) - keyword is necessary to accept Java clients.

  All other settings like compression and encryption depend on your needs. Please refer to "Configuration for channels running at the MQ server site" on page 11

6.3.1 Sample - Enabling NCI Java Security

The sample listed below only shows the code that is needed to connect to the queue manager. Because this is the part where the security is established.

Sample Java code

```
import java.util.*;
import com.ibm.mq.*;
import com.tsystems.nci.secmq.*;

class MQSample
{
public MQSample() {
super();
}

public static void main(java.lang.String[] args) {
Hashtable           connectProperties = new Hashtable(6);
NciMQSecurityExit   nciSecExit = null;

 try
 {
   // set proprties for MQSeries client connection
   connectProperties.put(MQC.HOST_NAME_PROPERTY,"53.113.127.88");
   connectProperties.put(MQC.CHANNEL_PROPERTY,"MQ01.CLIENT.CHL");
   connectProperties.put(MQC.PORT_PROPERTY,new Integer(1414));

   // instanciate NCI MQSeries security
   nciSecExit = new NciMQSecurityExit();
   nciSecExit.setUserId("bond");
   nciSecExit.setPwd("007");
   nciSecExit.setDataCompress("HUFF");

   // enable MQSeries security
   connectProperties.put(MQC.SECURITY_EXIT_PROPERTY,nciSecExit);
   connectProperties.put(MQC.SEND_EXIT_PROPERTY,nciSecExit);
   connectProperties.put(MQC.RECEIVE_EXIT_PROPERTY,nciSecExit);

   // connect to queue manager
   MQQueueManager qManager =
     new MQQueueManager("MQ01",connectProperties);
 }
 catch (MQException e)
 {
   if (nciSecExit.getErrorReasonCode() != 0)
   {
      // Connection error was caused by NCI MQSeries exits
      System.out.println("Error occured while MQ connection "
                         nciSecExit.getErrorMsgText());
   }
   else
   {
     // error caused by MQSeries
     e.printStackTrace();
   }
}

}
}
```

Sample channel definition - Platform z/OS

```
define  channel (MQ01.CLIENT.CHL) chltype (svrconn) trptype (tcp) +
        mcauser('USER1') +
        scyexit('NCIMCX') +
        scydata('HLQ.MQ01.INPUT(EXITCONF)') +
        sendexit('NCIMCX') +
        rcvexit('NCIMCX') +
        descr('Secure connection for MQSeries Java clients') +
        replace
```

Sample Exit Definition File

```
************************************
* Configuration for NCI MQSeries     *
* Channel Exits.                     *
************************************

...

ChannelName(MQ01.CLIENT.CHL)
  ClientAuth(YES)
  ClientAuthMethod(OS)
  EffectiveUser(PROPAGATE)
  EncryptionType(NONE)
  CompressionType(NONE)
  CompatMode(YES)
  TraceMode(ALL)

...
```

7 Appendix A. Sample exit configuration file

```
************************************
* ChannelExit  configuration       *
* QueueManager SAZ1MQCE            *
************************************

TraceFile(/var/mqm/qmgrs/SAZ1MQCE/errors/ncimcx.err)

LibraryName(ncimcx)
LibraryEntry(ChannelExit)

ScyExit(DYNAMIC)
SendExit(DYNAMIC)
RcvExit(DYNAMIC)

ScyData(/var/mqm/exits/SAZ1MQCE.edf)

*
* Sender channel (distributed queuing)
*
ChannelName(SAZ1MQ12.SAZ1MQ34)
  EncryptionType(DES)
  EncryptionKey(0x7465666768693434)
  CompressionType(LZ)
  TraceMode(INFO)
*
* Receiver channel (distributed queuing)
*
ChannelName(SAZ1MQ34.SAZ1MQ12)
  EncryptionType(DES)
  EncryptionKey(0x7465666768693434)
  CompressionType(LZ)
  TraceMode(INFO)
*
* Server-connection channel
*
ChannelName(SAZ1MQ34.SAZ1MQ12)
  ClientAuth(YES)
  ClientAuthMethod(OS)
  EffectiveUser(PROPAGATE)
  EncryptionType(DES)
  EncryptionKey(0x7465666768693434)
  CompressionType(LZ)
  TraceMode(INFO)
*
* Cluster channel with dynamically assigned library names
* (for heterogenous (different OS's) cluster environment)
*
ChannelName(TO.SAZ1MQCE)
  EncryptionType(DES)
  EncryptionKey(0x7465666768693434)
  CompressionType(LZ)
  TraceMode(NONE)

*
* Cluster channel with fixed library names
* (for homogenous (only one OS) cluster environment)
*
ChannelName(TO.SAZ1MQDE)
  ScyExit(/somewhere/libncimcx.sl(ChannelExit))
  SendExit(/somewhere/libncimcx.sl(ChannelExit))
  RcvExit(/somewhere/libncimcx.sl(ChannelExit))
  EncryptionType(DES)
  EncryptionKey(0x6465666768693435)
  CompressionType(LZ)
  TraceMode(NONE)
```

Index

Backpage

T-Systems Enterprise Services GmbH
Computing Services & Solutions (CSS)
System Products & Automation (MSY-PA)
Fasanenweg 9, 70771 Leinfelden-Echterdingen, Germany

Phone : +49 89/1011-4687
Fax. : +49 711/972-91622
E-mail : cc.middleware@t-systems.com
Internet : http://www.t-systems-systemproducts.com