


[print](#) | [close](#)

## APIs by Example: Cryptographic Key Management - Loading and Setting Master Keys

[System iNetwork Programming Tips Newsletter](#)

[Carsten Flensburg](#)

Carsten Flensburg

Thu, 11/08/2007 (All day)

In a number of previous articles I have discussed the Cryptographic Services APIs and provided commands and code to support a cryptographic infrastructure, including key management facilities. However, release V5R4 has introduced many new cryptographic APIs to ease and support the key management part of the challenge involved in setting up a secure and manageable cryptographic environment.

In today's issue of APIs by Example, I provide CL command interfaces to some of these APIs, with the intention of extending their immediate employment and usefulness, as well as discussing the API functions and the implications of their use. With that in mind, here are the Load Master Key Part (LODMSTKP) and Set Master Key (SETMSTK) commands.

The built-in support of cryptographic key management provided by release V5R4 provides the facilities required to establish a key hierarchy. In my previous articles on this topic I have discussed the purpose of a key hierarchy in detail, but here's a brief recap of the basics of a three-tier hierarchical key management system:

1. Encrypting sensitive data only protects the encrypted data as long as the data encryption key is not compromised or revealed. Fundamentally this is no different than the object security we commonly rely on when setting up data protection; if you can get at the key, you can get at the data. So obviously you will want to avoid storing data encryption keys in clear text.
2. A key encryption key (KEK) secures the data encryption key. Each KEK can be used to encrypt a group of data encryption keys. To reduce the risk and exposure in case of a KEK being lost or compromised, it makes good sense not to protect all data keys under the same KEK.
3. To protect the KEK and further reduce the number of keys to keep secret, the KEK is encrypted under a master key, the top level in the key hierarchy. Now your only challenge is to keep the master key secret, and as you will see in a moment, IBM has employed a pretty simple and yet ingenious method of facilitating that requirement.

For a more detailed explanation, see the aforementioned Cryptographic Services APIs articles (links to these articles are found below).

As of V5R4, the System i is capable of storing and maintaining a total of eight master keys. These eight master keys are stored in a location, not accessibly by any other external interface than the Cryptographic Services APIs. Each master key is a 256-bit Advanced Encryption Standard (AES) key stored as a 32-byte value in the License Internal Code (LIC) area, and has the following noteworthy properties and aspects:

- Each of the eight master keys exists in three versions: new, current, and old. The new version contains the already entered part(s) of a new master key currently being composed. The current version is the currently active master key. The old version contains the previous current version of the master key, and ensures that keys encrypted under this version of the master key are still recoverable.
- A master key is created from one or more master key parts, loaded individually as passphrases (text strings). Each passphrase is hashed using the SHA-512 hash algorithm to obtain a 32-byte hash value. The hash value is then exclusive or'd (XOR) with the current value of the new version of the master key.
- This process is continued until all key parts have been entered. The sequence in which the passphrases are entered does not have any significance. At that point, the Set Master Key API applies a final scramble process to the new key version and moves the resulting master key to the current key version, as well as the previous current key version to the old version. The previous old key version is removed from the system.
- When a current key version has been replaced, you should immediately re-encrypt (translate) all encryption keys encrypted under the (now) old version. Various APIs exist to support that requirement. But to help the cryptographic functions identify the correct key version until all encryption keys have been translated, a 20-byte key verification value (KVV) is calculated and associated with the master key, when it is set.
- When a key is encrypted under a master key and stored in a system key store file, the KVV of the master key is stored in the key record along with the key value, enabling the system to locate the correct master key version automatically. If you store a key encrypted under a master key outside of a system key store, you should ensure that the KVV is stored with it as well.
- If you perform a cryptographic function using the old master key version, a diagnostic message conveying the need to translate the key is issued to both the cryptographic API in question and the QSYSOPR message queue.
- The master keys are not included in any system save operations. The only way to restore a lost master key, for example in the event of a system recovery, is by re-entering the exact same passphrase(s) as initially entered when the master key was created. Since identical passphrases result in identical KVV's, you can also use the KVV to verify that the master key has been properly restored.
- Each of the Master Key APIs -- Load Master Key Part, Set Master Key, Clear Master Key, and Test Master Key -- creates a security audit record of type CY with a detailed entry value of M (Master Key function).

Care and thought is obviously required when deciding how the master key maintenance procedure should be conducted to establish a safe and secure key management setup. Of course, it is also very important to properly devise, carefully document, and promptly communicate the procedure to all parties involved.

Using a hierarchical key management system lets you create each required master key from 2, 3, or more passphrases entered by as many individuals, so that no single person has the knowledge required to produce (or recover) the master key. Since all other key encryption and data keys are encrypted, this solution definitely provides a healthy foundation for implementing a cryptographic

key infrastructure and management procedure. From there on, it's up to the individual shop to perform the appropriate planning and required programming to actually succeed in meeting the challenge.

To narrow that gap and shorten the jump a bit for you, I've written a couple of CL commands implementing the Load Master Key Part and Set Master Key functions based on the equivalent APIs. Here's the LODMSTK command prompt:

```

                                Load Master Key Part (LODMSTKP)

Type choices, press Enter.

Master key ID . . . . . 1-8

Passphrase . . . . .

Passphrase (to verify) . . . . .

```

Specify the master key ID for the master key you want to load a master key part for, enter the passphrase that should be hashed and added to the current value of the new master key. Repeat the passphrase to verify it and avoid spelling errors or other mistakes that could make it impossible to re-create the master key, should the need arise. Refer to the command help text to learn more about the LODMSTKP command.

Once you have entered all master key parts, the next step is to finalize the setting of the master key. As mentioned earlier, this step will remove the old version of the specified master key, move the current version to the old version, and the new version to current version. Following this event, the new version will be reset to all x'00'. Fully prompted, the SETMSTK command has the following appearance:

```

                                Set Master Key (SETMSTK)

Type choices, press Enter.

Master key ID . . . . . 1-8

Output . . . . . *          *, *PRINT, *STMF,
*MSG
Key verification value file . .

```

Again, you specify the ID of the master key to set. If no key parts have previously been added, an error message will be returned. The output parameter defines where the KVV associated with the new master key will be returned. The following options apply:

```

*          Displays the 20 byte KVV value in hexadecimal format in a
display panel. Due to the
           formatting the KVV will occupy 40 bytes. From the display

```

```

panel function key F21 allows
    you to print the information.

*PRINT      Prints the 20 byte KVV value in hexadecimal format to a
spooled file and places
    the file in the job's current default output queue. Due to
the formatting taking place
    the KVV will occupy 40 bytes.

*STMF       Writes the 20 byte KVV to stream file specified in the key
verification value file path.

*MSG        Returns the 20 byte KVV as message data in a completion
message sent to the caller
    of the command. The message will display the KVV in
hexadecimal format, while the message
    data remains unformatted.

```

Here's an example of how the display panel would look following a successful setting of master key 1:

```

                                     Set Master Key

WYNDHAMW                                     04-11-07

20:26:06
Master key ID . . . . . :    1
Key verification value . . :
C771ED68C7F78F26818E64E986289E67184A01BF
Press Enter to continue

F3=Exit   F12=Cancel   F21=Print master key information

Master key 1 has been successfully set.

```

Note that the key verification value has been formatted to 40-hex nibbles to display a recognizable value. In reality the KVV is received as a 20-byte binary string. Again, the command and display panel help text reveals all the details.

The Load Master Key Part and Set Master Key APIs both require \*ALLOBJ and \*SECADM special authority to run successfully. For this reason, the service program calling these APIs on behalf of the equivalent CL commands is configured to adopt user profile QSECOFR's authority. Instead, the CL commands are restricted from being run by unauthorized users by means of function usage authorization. This topic has also previously been covered in this column, and links to the relevant articles are provided below, should you be interested in refreshing your memory.

The CBX180M CL program provided to build the LODMSTKP and SETMSTK commands registers a special and individual user function for the LODMSTKP and SETMSTK commands, respectively. The user running the CBX180M CL program will be authorized to both commands. Follow the instructions specified below to create all command objects correctly. Use the following command to change the two function usage registrations in accordance with your requirements:

```
WRKFCNUSG FCNID(CBX_CRYPTO_MASTERKEY*)
```

Be sure that any person given access to the LODMSTKP and SETMSTK commands fully comprehends the scope and possible consequences of loading and setting master keys before they are set loose -- here's an excerpt from the two commands' help text aimed at preventing that kind of mistake from happening:

- Do not attempt to run these commands unless you have been explicitly authorized to do so by the proper authority in your organization.
- Do not attempt to run these commands unless you have received thorough instructions in performing this function.
- Failing to comply with the above recommendations could lead to a major loss of critical production data.

If you are interested in reading more about key management, master keys, and related utilities, look for upcoming issues of APIs by Example.

Thank you to Beth Hagemester of IBM for a very helpful support in my research for this article.

**This APIs by Example includes the following sources:**

```
CBX180  -- RPGLE  -- Cryptographic Key Management - Services
CBX180B -- SRVSRC -- Cryptographic Key Management - Binder source

CBX181  -- RPGLE  -- Load Master Key Part - CPP
CBX181H -- PNLGRP -- Load Master Key Part - Help
CBX181X -- CMD    -- Load Master Key Part

CBX182  -- RPGLE  -- Set Master Key - CPP
CBX182E -- RPGLE  -- Set Master Key - UIM Exit Program
CBX182H -- PNLGRP -- Set Master Key - Help
CBX182P -- PNLGRP -- Set Master Key - Panel Group
CBX182V -- RPGLE  -- Set Master Key - VCP
CBX182X -- CMD    -- Set Master Key

CBX180M -- CLP    -- Cryptographic Key Management - Build commands
```

To create all above objects, compile and run CBX180M. Compilation instructions are found in the source headers as usual. Note that the two previously published commands -- Add Function Registration (ADDFCNREG) and Change User Function Usage (CHGUSRFCNU)-- are required for the master key commands to run successfully and the CBX180M program to compile.

The sources for the two user function commands are included with this article. Links to the articles explaining these commands in great detail are located at the end of this article. The following sources are involved:

```
CBX1401  -- RPGLE  -- Add Function Registration - CPP
CBX1401H -- PNLGRP -- Add Function Registration - Help
CBX1401O -- RPGLE  -- Add Function Registration - POP
CBX1401V -- RPGLE  -- Add Function Registration - VCP
CBX1401X -- CMD    -- Add Function Registration

CBX141   -- RPGLE  -- Change User Function Usage - CPP
```

```
CBX141H  -- PNLGRP  -- Change User Function Usage - Help
CBX141O  -- RPGLE   -- Change User Function Usage - POP
CBX141X  -- CMD     -- Change User Function Usage
```

To create all above objects please follow the compilation instructions in the respective source headers.

### Previously published related articles:

Cryptographic Services APIs: Key Management:

<http://www.systeminetwork.com/article.cfm?id=20470>

APIs by Example: Cryptographic Services APIs, Part 1:

<http://www.systeminetwork.com/article.cfm?id=51236>

APIs by Example: Cryptographic Services APIs, Part 2:

<http://www.systeminetwork.com/article.cfm?id=51786>

APIs by Example: Cryptographic Services APIs, Part 3:

<http://www.systeminetwork.com/article.cfm?id=51863>

APIs by Example: Cryptographic Services APIs, Part 4:

<http://www.systeminetwork.com/article.cfm?id=51962>

APIs by Example: Cryptographic Services APIs, Part 5:

<http://www.systeminetwork.com/article.cfm?id=52017>

APIs by Example: Cryptographic Services APIs, Part 6:

<http://www.systeminetwork.com/article.cfm?id=52119>

APIs by Example: Cryptographic Services APIs, Part 7:

<http://www.systeminetwork.com/article.cfm?id=52224>

APIs by Example: User Function Registration APIs, Part 1:

<http://www.systeminetwork.com/article.cfm?id=51361>

APIs by Example: User Function Registration APIs, Part 2:

<http://www.systeminetwork.com/article.cfm?id=51418>

APIs by Example: User Function Registration APIs, Part 3:

<http://www.systeminetwork.com/article.cfm?id=51525>

### Other related documentation:

IBM -- Cryptographic Services Master Keys:

<http://publib.boulder.ibm.com/infocenter/iserics/v5r4/topic/apis/qc3MasterKeys.htm>

Wikipedia -- Key Management:

[http://en.wikipedia.org/wiki/Key\\_management](http://en.wikipedia.org/wiki/Key_management)

Wikipedia -- SHA Hash Functions:

<http://en.wikipedia.org/wiki/SHA-1>

### This article demonstrates the following Cryptographic Services API:

Load Master Key Part (Qc3LoadMasterKeyPart) API:

<http://publib.boulder.ibm.com/infocenter/iserics/v5r4/topic/apis/qc3ldmkp.htm>

Set Master Key (Qc3SetMasterKey) API:

<http://publib.boulder.ibm.com/infocenter/iserics/v5r4/topic/apis/qc3setmk.htm>

Key Management APIs:

<http://publib.boulder.ibm.com/infocenter/iserics/v5r4/topic/apis/catcrypt6.htm>

Cryptographic Services APIs:

<http://publib.boulder.ibm.com/infocenter/iserics/v5r4/topic/apis/catcrypt.htm>

**You can retrieve the source code for this API example from:**

[http://www.pentontech.com/IBMContent/Documents/article/55862\\_410\\_MasterKeys.zip](http://www.pentontech.com/IBMContent/Documents/article/55862_410_MasterKeys.zip).

**Source URL:** <http://iprodeveloper.com/rpg-programming/apis-example-cryptographic-key-management-loading-and-setting-master-keys>