

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

APIs by Example: Cryptographic Services APIs, Part 4

[System iNetwork Programming Tips Newsletter](#)

[Carsten Flensburg](#)

Carsten Flensburg

Thu, 01/12/2006 (All day)

In part 3 of this article series, I delivered the Create Master Key (CRTMSTK) and Remove Master Key (RMVMSTK) commands. This time, I add the Create Key Encrypting Key (CRTKEK) and Remove Key Encrypting Key (RMVKEK) commands to the set of key administration tools that I intend to offer as part of my exploration of the Cryptographic Services APIs.

As explained last time, the CRTMSTK command simply adds the specified master key to the key store as it is, whereas the CRTKEK command needs to encrypt the specified key encrypting key under the master key before storing it. It is here that one of the new improvements to the Cryptographic Services APIs comes into play: key context tokens.

Instead of retrieving the master key and passing that on in clear text to the encryption process, a key context token is returned from the function responsible for retrieving the master key. A key context token identifies the actual key value to the encryption and decryption APIs, and this information is stored below the MI and therefore accessible only to system functions. Passing the token instead of the key itself between functions that require the master key reduces the risk of exposure and thereby the risk of the master key being compromised.

A key context token is valid only in the job that created it, and it cannot be passed from one job to another. Likewise, an algorithm context token can be created to point to the algorithm parameters used in the encryption and decryption process, such as cipher algorithm, block length, mode, pad option, and so on. I use algorithm context tokens in this example because they centralize and thereby simplify the control of the algorithm parameters in the cryptographic processes.

The CRTKEK command performs the following steps:

1. An algorithm context token is created.
2. A master key context token is created.
3. If the special value *GEN was specified for the key value, the Generate Symmetric Key (QC3GENSK, Qc3GenSymmetricKey) API is called, passing the master key context token to let the API return the new key already encrypted under the master key.
4. If a key value was specified, the Encrypt Data (QC3ENCDDT, Qc3EncryptData) API is called, passing the key value and master key context token.
5. The encrypted key-encrypting key (KEK) resulting from step 3 or 4 is stored in the key store validation list.
6. The context tokens are destroyed.

Here's what the prompt of the Create Key Encrypting Key command looks like:



```

                                Create Key Encrypting Key (CRTKEK)
Type choices, press Enter.
Key label . . . . .
Key length . . . . . 16          16, 24, 32
Key bytes 1-8 . . . . . *GEN
Key bytes 9-16 . . . . . *GEN

```

The key label parameter is the name or identifier of the KEK. The specified KEK label is used to identify by which KEK a data encryption key is encrypted, later in the key management process.

I provide a help panel group for both the CRTKEK and RMVKEK commands, to explain all the command parameters in detail.

In the next installment of this series, I will complete the command set necessary to create or remove a master key, KEKs, and data encryption keys and present the Create Data Key (CRTDTAK) and Remove Data Key (RMVDTAK) commands. In a later installment, I will also offer tools to change a master key or KEK and at the same time re-encrypt any of its sub-keys under the new key.

Note that for simplicity I store all encryption keys in the same key store, but if necessary, I could use different key stores to further isolate the access to the different keys and key types. As for the protection issue in a broader perspective, I must emphasize that encryption of the cipher keys is only one among more protection mechanisms, which, when added together, should provide an assessed and acceptable level of risk, as far as unauthorized access to the cipher keys, as well as the data they are protecting, is concerned.

Other aspects of data protection and key management include such things as object authority, audit journal and control, program design, and risk management in general, as well as developing and enforcing a company security policy. You must take into account all these different aspects individually for each application, depending on the specific requirements and risk exposures of that application.

This key management sample application is mainly intended as an introduction to the Cryptographic Services APIs and as a starting point for anyone faced with the need to develop cryptographic applications. And as I mentioned, careful research and design efforts are mandatory to ensure that any cryptographic application that you put into production is in accordance with the specific requirements and development guidelines of your shop.

Encryption and decryption in this key management sample application are performed using the Cryptographic Services APIs implementation of the Advanced Encryption Standard (AES) block cipher algorithm and a block size of 16 bytes.

The following cryptographic and key management functions are available with this article and the previous articles in this series:

```

GenAesKey() -- Generate AES cipher key
GenInzVct() -- Generate initialization vector
GetAlgCtx() -- Get algorithm context
GetMgtAlg() -- Get key management algorithm context
GetKeyCtx() -- Get key context
RmvAlgCtx() -- Remove algorithm context
RmvKeyCtx() -- Remove key context
EncDtaStr() -- Encrypt data string using context tokens

```

```
DecCphStr() -- Decrypt cipher string using context tokens

AddKeyEnt() -- Add key entry to key store
ChgKeyEnt() -- Change key store entry
ChkSubKey() -- Check sub key existence
FndNxtKeyE() -- Find next key entry
FndTopKeyE() -- Find top key entry
GetKeyAtr() -- Get key attribute
GetKeySto() -- Get key store
GetMstKeyLb() -- Get master key label
RmvKeyEnt() -- Remove key store entry
VfyKeyEnt() -- Verify key store entry
GetFcnUsq() -- Get function usage
GetMstKeyTk() -- Get master key context token
```

I will continue the coverage of the V5R3 Cryptographic Services APIs in the coming APIs by Example columns and in the course of that process add to the preceding list of cryptographic and key management functions.

You can find part one of this article here:

<http://www2.systeminetwork.com/article.cfm?id=51236>

You can find part two of this article here:

<http://www2.systeminetwork.com/article.cfm?id=51786>

You can find part three of this article here:

<http://www2.systeminetwork.com/article.cfm?id=51863>

This APIs by Example includes the following sources:

```
CBX146 -- Cryptographic services service program
CBX146B -- Service program binder source
CBX147 -- Cryptographic key management service program
CBX147B -- Service program binder source

CBX1471 -- Create Master Key - command processing program
CBX1472 -- Remove Master Key - command processing program
```

The preceding sources are all revised versions of previously published sources, which I've updated to support requirements introduced with this article. Please replace these sources in your utility library's source files with these updated versions. The CBX148M program ensures that the program objects get correctly recompiled.

The following new sources deliver the CRTKEK and RMVKEK commands:

```
CBX1481 -- Create Key Encrypting Key - command processing program
CBX1481H -- Create Key Encrypting Key - help
CBX1481V -- Create Key Encrypting Key - validity checker
CBX1481X -- Create Key Encrypting key - command

CBX1482 -- Remove Key Encrypting Key - command processing program
CBX1482H -- Remove Key Encrypting Key - help
```

```
CBX1482V -- Remove Key Encrypting Key - validity checker
CBX1482X -- Remove Key Encrypting Key - command
```

I also include a program that performs all necessary command object creation:

```
CBX148M -- Command objects creation
```

Compilation instructions are also in the source headers, as usual.

This article demonstrates the following APIs:

Add Validation List Entry (QsyAddValidationLstEntry) API:

<http://publib.boulder.ibm.com/infocenter/iserics/v5r3/topic/apis/qsyavle.htm>

Change Validation List Entry (QsyChangeValidationLstEntry) API:

<http://publib.boulder.ibm.com/infocenter/iserics/v5r3/topic/apis/QSYCVLE.htm>

Find First Validation List Entry (QsyFindFirstValidationLstEntry) API:

<http://publib.boulder.ibm.com/infocenter/iserics/v5r3/topic/apis/QSYFFVLE.htm>

Find Next Validation List Entry (QsyFindNextValidationLstEntry) API:

<http://publib.boulder.ibm.com/infocenter/iserics/v5r3/topic/apis/QSYFNVLE.htm>

Find Validation List Entry (QsyFindValidationLstEntry) API:

<http://publib.boulder.ibm.com/infocenter/iserics/v5r3/topic/apis/QSYFIVLE.htm>

Remove Validation List Entry (QsyRemoveValidationLstEntry) API:

<http://publib.boulder.ibm.com/infocenter/iserics/v5r3/topic/apis/QSYRVLE.htm>

Send Program Message (QMHSNDPM) API:

<http://publib.boulder.ibm.com/infocenter/iserics/v5r3/topic/apis/QMHSNDPM.htm>

Move Program Messages (QMHMOVPM) API:

<http://publib.boulder.ibm.com/infocenter/iserics/v5r3/topic/apis/qhmovpm.htm>

Resend Escape Message (QMHRSNEM) API:

<http://publib.boulder.ibm.com/infocenter/iserics/v5r3/topic/apis/QMHRSNEM.htm>

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

http://www.pentontech.com/IBMContent/Documents/article/51962_50_CryptoServices4.zip

Source URL: <http://iprodeveloper.com/rpg-programming/apis-example-cryptographic-services-apis-part-4>