

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

APIs by Example: Cryptographic Services APIs, Part 5

[System iNetwork Programming Tips Newsletter](#)

[Carsten Flensburg](#)

Carsten Flensburg

Thu, 01/26/2006 (All day)

This installment of APIs by Example focuses on the tools required to create and remove data encryption keys: the Create Data Encryption Key (CRTDTAK) and Remove Data Encryption Key (RMVDTAK) commands, respectively. As its name suggests, the data encryption key is the cipher key used to perform the actual encryption of the cleartext string to be encrypted.

In the next installment of this series, I will show you how to use functions (which I'll be providing) to successfully complete the cleartext encryption and ciphertext decryption process, using a data encryption key.

For now, I'll continue with a very important warning concerning data encryption keys: Removing or destroying a data encryption key removes your access to the data encrypted with it as well. Consequently, if you cannot restore the data encryption key, you cannot restore your access to the encrypted data, if any. A secure backup policy for the key store is therefore mandatory.

The same warning of course applies to the master key and the key-encrypting keys (KEKs). As long as you manage these with the tools that I provide with this article, however, there should be no risk of accidentally removing these keys. Both the Remove Master Key (RMVMSTK) and the Remove Key Encrypting Key (RMVKEK) commands check whether any lower-level keys are encrypted with the key that they are about to remove, before doing so.

The details about implementing cryptographic applications in general are beyond this article's scope, but if you're facing such a requirement, I would like to draw your attention to a recent iSeries Network Webcast, "Practical Tips for Implementing Encryption," featuring security expert and author Carol Woodbury and iSeries expert Jon Paris. This Webcast offers a wealth of information about how to understand and surmount the challenges involved in analyzing and determining the scope of such a project, as well as how to set up secure cryptographic environments and program related applications.

[Click here to view the recorded Webcast \(registration required\)](#)

Further, for a broad perspective on the many and different aspects involved in a secure implementation of cryptographic applications, I also recommend Bruce Schneier's essay "Security Pitfalls in Cryptography": <http://www.schneier.com/essay-028.html>

And now it's time to have a look at the data encryption key commands: Similar to the Create Key Encrypting Key (CRTKEK) command that I provided last time and which stored the specified KEK encrypted under the master key, the CRTDTAK command also stores the specified data encryption encrypted. At this level, the KEK specified on the CRTDTAK command is used as an encryption key.

The CRTDTAK command performs the following steps:

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

Please note that to retrieve the KEK context token in step two, a master key context token is retrieved by the function returning the KEK context token. To retrieve a master key context token, a special usage authorization is required, as I explained in part three of this series.

Here's what the CRTDTAK command prompt looks like:

```

                                Create Data Encryption Key (CRTDTAK)
Type choices, press Enter.
Key label      . . . . .
Key encryption 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. Later in the key extraction process, the specified KEK label is also used to identify by which KEK a data encryption key is encrypted.

To explain all the command parameters in detail, I provide a help panel group for both the CRTDTAK and RMVDTAK commands.

Having built all the key management commands published so far, you should now be able to establish a full encryption key hierarchy, as in the following example, that lets the system generate the key values:

1. Run the command CRTMSTK
2. Run the command CRTKEK KEYLABEL(CBX_KEY_0001)
3. Run the command CRTDTAK KEYLABEL(CBX_DTAK_0001) KEKLABEL(CBX_KEY_0001)

To be able to run the key management commands above, a user must have usage authorization to the CBX_CRYPTO_KEY_MANAGEMENT user function. The command WRKFCNUSG FCNID (CBX_CRYPTO_KEY_MANAGEMENT) shows you exactly which user profiles are authorized and also lets you add or remove user profiles.

Following a successful execution of the preceding commands, you will have a fully operational and encrypted data encryption key in your key store. In the next installment of this series, I build a sample application that shows how to use this data encryption key to encrypt and decrypt sensitive business data.

I've provided the following cryptographic and key management functions with this and 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
GetFcnUsg() -- Get function usage
GetMstKeyTk() -- Get master key context token
GetKekTkn() -- Get key encryption 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 this list of cryptographic and key management functions.

NOTE: Before using any or part of the tools I provide in this article series in a production environment, I recommend reading all parts of this series and taking into account all recommendations and warnings stated in each part of this series.

You can find part one of this article here:

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

Part two here:

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

Part three here:

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

Part four here:

<http://www2.systeminetwork.com/article.cfm?id=51962>

This APIs by Example includes the following source members:

```

CBX147 -- Cryptographic key management service program
CBX147B -- Service program binder source
CBX1481H -- Create Key Encryption Key - help

```

The preceding sources are all revised versions of previously published sources and have been updated to support new functions introduced in this article as well as to correct an error in the help

panel group. Please replace these sources in your utility library's source files. The CBX149M program ensures that the service program gets correctly recompiled.

The following new sources deliver the CRTDTAK and RMVDTAK commands:

```
CBX1491 -- Create Data Encryption Key - command processor
CBX1491H -- Create Data Encryption Key - help
CBX1491V -- Create Data Encryption Key - validity checker
CBX1491X -- Create Data Encryption key - command

CBX1492 -- Remove Data Encryption Key - command processor
CBX1492H -- Remove Data Encryption Key - help
CBX1492V -- Remove Data Encryption Key - validity checker
CBX1492X -- Remove Data Encryption Key - command
```

I have included a program that performs all necessary command object creation:

```
CBX149M -- 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>

Encrypt Data (Qc3EncryptData) API:

<http://publib.boulder.ibm.com/infocenter/iserics/v5r3/topic/apis/qc3encdt.htm>

Decrypt data (Qc3DecryptData) API:

<http://publib.boulder.ibm.com/infocenter/iserics/v5r3/topic/apis/qc3decdt.htm>

Generate Symmetric Key (Qc3GenSymmetricKey) API:

<http://publib.boulder.ibm.com/infocenter/iserics/v5r3/topic/apis/qc3gensk.htm>

Generate Pseudorandom Numbers (Qc3GenPRNs) API:

<http://publib.boulder.ibm.com/infocenter/iserics/v5r3/topic/apis/qc3genprns.htm>

Create Algorithm Context (Qc3CreateAlgorithmContext) API:

<http://publib.boulder.ibm.com/infocenter/iserics/v5r3/topic/apis/qc3crtax.htm>

Create Key Context (Qc3CreateKeyContext) API:

<http://publib.boulder.ibm.com/infocenter/iserics/v5r3/topic/apis/qc3crtkx.htm>

Destroy Algorithm Context (Qc3DestroyAlgorithmContext) API:

<http://publib.boulder.ibm.com/infocenter/iserics/v5r3/topic/apis/qc3desax.htm>

Destroy Key Context (Qc3DestroyKeyContext) API:

<http://publib.boulder.ibm.com/infocenter/iserics/v5r3/topic/apis/qc3deskx.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/qmhmovpm.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/52017_53_CryptoServices5.zip

Source URL: <http://iprodeveloper.com/rpg-programming/apis-example-cryptographic-services-apis-part-5>