


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

## APIs by Example: Cryptographic Key Management - Creating and Translating Key Stores

[System iNetwork Programming Tips Newsletter](#)

[Carsten Flensburg](#)

Carsten Flensburg

Thu, 01/24/2008 (All day)

As of V5R4, i5/OS now includes key stores. Technically, key stores have an object type of \*FILE -- and more specifically, a physical data base file -- but data access to key stores is only possible through the Cryptographic Services APIs. Key stores offer the option of storing key encryption keys or data keys, securely encrypted under a master key. If you are unfamiliar with the concepts of master keys, key encryption keys, and data keys, please see my earlier Cryptographic Key Management articles, which cover those topics in more detail; links are provided at the end of this article.

The Cryptographic Services APIs for managing key stores provide a variety of functions, such as creating a key store, generating or writing a key store record, and deleting a key store record. Other key store APIs let you retrieve the attributes of a key store record or translate all the key store records in a key store encrypted with one master key to another. To discuss key stores in more detail and provide some key store API coding examples, today's issue of APIs by Example presents a pair of CL commands named Create Key Store (CRTKS) and Translate Key Store (TRNKS) that are based on the respective APIs.

The Cryptographic Services APIs that require a key as input to a cryptographic operation have had added a new KEYDo400 key description parameter format to enable the use of key encryption keys or data keys stored in a key store. This new format lets you specify a qualified key store name and a key label for the key parameter. The key label is the name that uniquely identifies a specific key store record within a key store; you define it when you add the key record to the key store. I discuss key store records in more detail in a moment, and an upcoming APIs by Example will look more closely at the options and methods controlling the management of key store records.

Specifying the KEYDo400 format as input to a Cryptographic Services API causes the API to retrieve the key from the key store and decrypt it with the master key associated with the key store. While in the key store, the key is encrypted and thereby securely protected by the master key. The option of specifying the key store record directly to a Cryptographic Services API lets you maintain a closed circuit for the duration of the entire encryption process performed by that API, thus avoiding the exposure of the key outside of the API domain.

However, this circumstance requires careful consideration on your part to ensure that users are given proper authorization to both the key stores and the APIs involved. Anyone obtaining this access will be able to extract the key from the key store, or use it to retrieve the data encrypted under it. Likewise, it's very important that you include key stores in your backup schedule to enable recovery of the key encryption keys and data keys stored there. Imagine if you had a system failure, and restored your data to a new computer, only to learn that none of the encrypted data is accessible because the key stores are gone! Further backup considerations apply at the point where new key

store records have been added or the key store has been translated. I explain key store translation shortly, so bear with me if this is an unfamiliar term to you.

Key stores are a great addition to the Cryptographic Services APIs and the cryptographic infrastructure included in i5/OS -- let's look at how to create them with the Create Key Store (QC3CRTKS/Qc3CreateKeyStore) API. This API can be called as a program named QC3CRTKS, or as an ILE procedure named Qc3CreateKeyStore. For ease of use and to provide an example of how to call the API, I've created a Create Key Store (CRTKS) CL command -. here's what the CRTKS command prompt looks like:

Create Key Store (CRTKS)

Type choices, press Enter.

Key store . . . . .		Name
Library . . . . .	*CURLIB	Name, *CURLIB
Master key ID . . . . .		1-8
Text 'description' . . . . .	*BLANK	
Authority . . . . .	*EXCLUDE	Name, *EXCLUDE,
*LIBCRTAUT...		

You specify a qualified key store name, the Master Key ID of the master key you want to associate with the key store, as well as an optional text description and the public authority to assign to the key store. Please note that the specified Master Key ID need not be set at the point where the key store is created, but rather when key store records are actually added.

For the reasons explained earlier, I have defaulted the public authority I have to \*EXCLUDE. But remember that object authority offers no protection against user profiles that have \*ALLOBJ special authority in the profile! Careful planning of your user profile and object authorization scheme is still of utmost importance to ensure a safe and secure cryptographic infrastructure.

To try out the CRTKS utility, run the following command to create a key store by the name of KEYSTORE01 in the QGPL library:

```
CRTKS KEYSTORE(QGPL/KEYSTORE01)
      KEYID(7)
      TEXT(*BLANK)
      AUT(*EXCLUDE)
```

If the above CRTKS command completes successfully, you'll receive a completion message that verifies that the key store was created.

Message ID . . . . .	CBX0021	Severity . . . . .
00		
Message type . . . . .	Completion	

```

Date sent . . . . . : 19-01-08      Time sent . . . . . :
11:40:40
Message . . . . . : Key store KEYSTORE01 created in library QGPL.

Cause . . . . . : The key store KEYSTORE01 was successfully
created in
library QGPL. The master key ID 7 will be used to encrypt the
cryptographic
keys stored in the key store.

```

To verify that a key store was created as opposed to an ordinary physical file, try executing the Run Query (RUNQRY) command against it. You will receive the QRY2293 exception message: *Query cannot be run. See lower level messages.* Checking out the job log leads to the discovery of a CPF4234 diagnostic message preceding the QRY2293 exception message:

```

Message ID . . . . . : CPF4234      Severity . . . . . :
50
Message type . . . . . : Diagnostic

Date sent . . . . . : 19-01-08      Time sent . . . . . :
11:55:51
Message . . . . . : Input or output operations for member
KEYSTORE01 not
allowed.

Cause . . . . . : Member KEYSTORE01 file KEYSTORE01 in library
QGPL could
not be opened because it does not allow any input or output
operations. The
input and output operations allowed for member KEYSTORE01 are
contained in
the attributes specified for file KEYSTORE01 in library QGPL. The

attributes of the file can be displayed by using the DSPFD
command.

```

The Display File Description (DSPFD) explains the cause of the above CPF4234 diagnostic message. Note the bolded part of this excerpt of the DSPFD command output:

```

19-01-08      Display File Description

DSPFD Command Input

File . . . . . : FILE
KEYSTORE01
Library . . . . . : QGPL

```

Type of information . . . . .	: TYPE	*ALL
File attributes . . . . .	: FILEATR	*ALL
System . . . . .	: SYSTEM	*LCL
File Description Header		
File . . . . .	: FILE	
KEYSTORE01		
Library . . . . .	:	QGPL
Type of file . . . . .	:	Physical
File type . . . . .	: FILETYPE	*DATA
Auxiliary storage pool ID . . . . .	:	00001
Data Base File Attributes		
Externally described file . . . . .	:	Yes
SQL file type . . . . .	:	TABLE
File level identifier . . . . .	:	
1080119114039		
Creation date . . . . .	:	19-01-08
Text 'description' . . . . .	: TEXT	
Distributed file . . . . .	:	No
Partitioned SQL Table . . . . .	:	No
DBCS capable . . . . .	:	No
Maximum members . . . . .	: MAXMBRS	1
Number of constraints . . . . .	:	1
Number of triggers . . . . .	:	0
Number of members . . . . .	:	1
...		
Reuse deleted records . . . . .	: REUSEDLT	*YES
Coded character set identifier . . . . .	: CCSID	65535
<b>Allow read operation . . . . .</b>	:	<b>No</b>
<b>Allow write operation . . . . .</b>	:	<b>No</b>
<b>Allow update operation . . . . .</b>	: <b>ALWUPD</b>	<b>*NO</b>
<b>Allow delete operation . . . . .</b>	: <b>ALWDLT</b>	<b>*NO</b>
Record format level check . . . . .	: LVLCHK	*YES
Access path . . . . .	:	Keyed
Access path size . . . . .	: ACCPTHsiz	*MAX1TB
Access path logical page size . . . . .	: PAGESIZE	*KEYLEN
Maximum key length . . . . .	:	97
Maximum record length . . . . .	:	2556
...		

As you can see, none of the data access operations are allowed for key store files: Read, write, update and delete operations are all prohibited. None of these attributes, however, prevent you from displaying the key store record format. Using the Display File Fields (DSPFFD2) command (presented in an earlier APIs by Example article) reveals the following database record format layout:

```

File . . . . . :   KEYSTORE01           Record length . . :   2556

  Library . . . :    QGPL               Field count . . . :    10

Record format . . :   KEYSTORE01

File type . . . :   PF                  Include field . .

Access path . . . :   *KEYED UNIQUE      Include text . . .

Field      Data type  Buffer  Length  Dig  Dec  Key  Text
-----
KYLABEL    Char      1      97                1  A
RESRV1     Char      98      3
KEYTYPE    Binary    101     4    9    0
KEYSIZE    Binary    105     4    9    0
TIMEDATE   Char     109     4
KVV        Char     113    20
KYVARIANT  Binary    133     4    9    0
CHECKSUM    Binary    137     4    9    0
RESRV2     Char     141     2
TOKEN      Var Char   143   2414
  
```

The above record format applies to all i5/OS key store files and the attributes together define the information stored in a key store record. But more about key store records next time. If you want to delete the key store created earlier, you can do so using the Delete File (DLTF) command:

```
DLTF FILE(QGPL/KEYSTORE01)
```

The second command I present today, Translate Key Store (TRNKS), is also based on a corresponding API -- the Translate Key Store API is named QC3TRNKS and Qc3TranslateKeyStore for the OPM and ILE versions, respectively.

Translation in this context reflects the event that the keys in the key store are first decrypted using the master key that originally encrypted the keys and, in the same process, immediately following decryption the keys are again encrypted using another master key. The need to perform this re-encryption process could be prompted by one of two circumstances:

1. The same master key ID has changed version since the original encryption of the key store records. This occurs if, using the Set Master Key command or API, a new master key version has been promoted to the current version, and the original version has been saved to the old version. To avoid the possible loss of the old master key version, should someone repeat the aforementioned process, I recommend that you translate all keys still encrypted under the old version of the master key to the current version as soon as possible, after you set the master key.
2. Suppose you decide to change the master key ID associated with a key store. Say a key store is currently associated with master key ID 1 and you need to change the key store master key to master key ID 2. You can use the TRNKS command or API to do this. The same precautions mentioned above apply here as well.

As you can see below, the Translate Key Store (TRNKS) command has a quite simple interface:

```

                                Translate Key Store (TRNKS)

Type choices, press Enter.
Key store . . . . . Name
Library . . . . . Name
          + for more values
Master key ID . . . . . 1-8

```

You can specify up to 32 different key stores in one execution of the TRNKS command. The TRNKS API will process each key store in turn, using the currently associated Master Key ID and the stored Key Verification Value (KVV) to locate the correct version of the master key (as explained last time) to decrypt all keys. Then, immediately following decryption, the current version of the specified master key ID is used to re-encrypt the keys. Should the translation process at some point encounter an error or exception, the process will be terminated immediately, and the job log of the executing job should be consulted to establish cause and consequences.

In addition to the security, confidentiality, and authorization aspects mentioned earlier, the following conditions and considerations apply to the utilities and APIs presented in this article: The Create Key Store API only requires the usual \*EXECUTE and \*ADD authority to the library in which the key store is created. I've decided that for my system though, to further add the requirement of \*ALLOBJ and \*SECADM special authority in the user profile executing the Create Key Store (CRTKS) command.

This precaution reduces the number of user profiles allowed to create key stores (using the CRTKS command) significantly on systems with an adequate security configuration, and I'd like to limit potential sources of errors and mistakes as much as possible, also in the context at hand. Since the source for the CRTKS CPP is included with this article, you're of course free to relax this restriction should you decide to do so.

The Translate Key Store API needs \*OBJOPR, \*READ and \*UPD authority to the key store object in order to process the key store records. I've replaced this requirement with function usage authorization. This implies that any user that attempts to run the Translate Key Store (TRNKS) command will need to be specifically and individually authorized to the CBX\_CRYPT0\_KEYSTORE\_XLATE that is registered by the CBX185M CL program provided with this article to build the CRTKS and TRNKS commands. The user running the CBX185M CL program will be authorized to the TRNKS command.

Use the following command to locate and change the function usage registrations applying to the key management utilities delivered with the Cryptographic Key Management articles in previous APIs by Example articles:

```
WRKFCNUSG FCNID(CBX_CRYPT0_*)
```

Given that you've loaded and installed the commands and usage registrations provided with this and the previous APIs by Example articles, you should see a list similar to the one below:

```

                                Work with Function Usage

Type options, press Enter.

    2=Change usage    5=Display usage

Opt  Function ID                                Function Name
translation
      CBX_CRYPT0_KEYSTORE_XLATE                  Cryptographic key store
      CBX_CRYPT0_MASTERKEY_CLEAR                 Clear cryptographic master key
load
      CBX_CRYPT0_MASTERKEY_LOAD                  Cryptographic master key part
      CBX_CRYPT0_MASTERKEY_SET                   Set cryptographic master key
      CBX_CRYPT0_MASTERKEY_TEST                  Cryptographic master key test

Bottom
Parameters for option 2 or command

===>

F3=Exit    F4=Prompt    F5=Refresh    F9=Retrieve    F12=Cancel
F17=Top
F18=Bottom

```

Use option 2 to add and/or remove users. function usage. You can find more information on function usage registration prerequisites and requirements in the APIs by Example article of December 13, 2007, please follow the link provided below.

I'll be continuing the Cryptographic Key Management coverage in upcoming issues of APIs by Example. Next time I'll show you how to generate, display, and delete key store records, so stay

tuned. And in the meantime, if you're facing the challenge of implementing cryptographic applications of your own and looking for information and inspiration, be sure to check out the links below pointing to NIST (National Institute of Standards and Technology) publications, providing a wealth of application security and cryptography related documentation and recommendations.

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

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

CBX185  -- RPGLE  -- Create Key Store - CPP
CBX185H -- PNLGRP -- Create Key Store - Help
CBX185X -- CMD    -- Create Key Store

CBX186  -- RPGLE  -- Translate Key Store - CPP
CBX186H -- PNLGRP -- Translate Key Store - Help
CBX186X -- CMD    -- Translate Key Store
CBX185M -- CLP    -- Cryptographic Key Management III - Build commands
```

To create all above objects, compile and run CBX185M. You'll find compilation instructions 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 Translate Key Store (TRNKS) command to run successfully - and for the CBX185M program to compile.

The sources for the two aforementioned user function commands were included with my previous APIs by Example article of November 8. In case you missed that article, you can find it at the link below. Successfully compiling and running the CBX180M CL setup program included with that article is a prerequisite to running the CBX185M setup program included today.

**Previously published related articles:**

APIs by Example: Working with Database Files, Fields and More:

<http://www2.systeminetwork.com/article.cfm?id=55705> (October 11, 2007)

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

<http://www2.systeminetwork.com/article.cfm?id=55862> (November 8, 2007)

APIs by Example: Cryptographic Key Management - Testing and Clearing Master Keys:

<http://www2.systeminetwork.com/article.cfm?id=56035> (December 13, 2007)

**Other related documentation:**

RFC 4107 Guidelines for Cryptographic Key Management:

<http://tools.ietf.org/html/rfc4107>

NIST (National Institute of Standards and Technology) Special Security Publications:

<http://csrc.nist.gov/publications/PubsSPs.html>

NIST SP 800-57: Recommendation for Key Management . Part 1:

[http://csrc.nist.gov/publications/nistpubs/800-57/sp800-57-Part1-revised2\\_Mar08-2007.pdf](http://csrc.nist.gov/publications/nistpubs/800-57/sp800-57-Part1-revised2_Mar08-2007.pdf)



NIST SP 800-57: Recommendation for Key Management . Part 2:

<http://csrc.nist.gov/publications/nistpubs/800-57/SP800-57-Part2.pdf>

NIST (National Institute of Standards and Technology) FIPS Publications:

<http://csrc.nist.gov/publications/PubsFIPS.html>

### **IBM documentation:**

i5/OS: Cryptography concepts:

<http://publib.boulder.ibm.com/infocenter/iserics/v5r4/topic/rzajc/rzajcconcepts.htm>

Cryptographic Services Master Keys:

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

Cryptographic Services Key Store:

<http://publib.boulder.ibm.com/infocenter/iserics/v5r4/topic/apis/qc3KeyStore.htm>

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

Create Key Store (Qc3CreateKeyStore) API:

<http://publib.boulder.ibm.com/infocenter/iserics/v5r4/topic/apis/qc3crtks.htm>

Translate Key Store (Qc3TranslateKeyStore) API:

<http://publib.boulder.ibm.com/infocenter/iserics/v5r4/topic/apis/qc3trnks.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>

[http://www.pentontech.com/IBMContent/Documents/article/56187\\_461\\_CrtKsTrnKs.zip](http://www.pentontech.com/IBMContent/Documents/article/56187_461_CrtKsTrnKs.zip).

**Source URL:** <http://iprodeveloper.com/rpg-programming/apis-example-cryptographic-key-management-creating-and-translating-key-stores>