

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

APIs by Example: Crypto Key Mgmt-Encrypt/Decrypt with Key Hierarchy

[System iNetwork Programming Tips Newsletter](#)

[Carsten Flensburg](#)

Carsten Flensburg

Thu, 04/24/2008 (All day)

If you have followed the APIs by Example articles covering cryptographic key management in general and the new key management APIs introduced with release V5R4 in particular, you should now be able to establish a three-tier cryptographic key hierarchy that lets you create and manage master keys, key encryption keys, and data keys. For anyone needing to read up on this exciting topic, I've provided links to all previous articles below.

One important part is still missing in this exercise, however: How to actually put this key hierarchy to practical use in a common application context. For this purpose, I have created a couple of CL commands that act as an interface to creating and changing customer data records stored in a physical data file: Add Customer Record (ADDCUSRCD) and Change Customer Record (CHGCUSRCD). The scenario prompting the encryption efforts is the following: One of the fields in the customer file contains information of a highly sensitive and confidential nature, and it is therefore required that the data stored in this field be stored in an encrypted format.

To begin today's journey, however, let me briefly describe the setup required to implement a three-tier cryptographic key hierarchy involving the new V5R4 key management facilities:

1. You load and set one or more (up to a maximum of 8) master keys, using the Load Master Key Part (LODMSTK) and the Set Master Key (SETMSTK) commands, respectively.
2. You create a key encryption key store using the Create Key Store (CRTKS) command, assigning the master key ID of the master key that the system will use to encrypt all keys as they are stored in the key store. Likewise, the system will use that master key ID to decrypt the keys when they are retrieved from the key store.
3. You generate and add a key encryption key to the newly created key store using the Generate Key Record (GENKR) command. To uniquely identify the key encryption key, you specify a key label name for the equivalent parameter on the GENKR command. The GENKR command also offers a variety of key attributes and properties to be defined, as dictated by your specific requirements.
4. You create a data key store using the Create Data Key Store (CRTDTAKS) command. You specify a key encryption key store, as well as a key encryption key label, referring to the key encryption key, that you want to be used for the encryption of all data keys in the data key store.
5. You add a data encryption key to the newly created data key store using the Create Data Key (CRTDTAK) command. Again you specify a key label name to identify the data key record as

well as the key length. The Advanced Encryption Standard (AES) algorithm is used for all key encryption key and data encryption key operations performed by the CRTDTAK command.

At the end of this article, I provide links to the previous APIs by Example articles introducing and delivering the commands mentioned above. Here's an example of how the above prerequisite set of commands (2-5) could be executed; I've placed the key stores in library QGPL, but you can of course change that to whatever library you prefer:

```

2. CRTKS KEYSTORE (QGPL/CBX192)
   KEYID (1)

3. GENKR KEYSTORE (QGPL/CBX192)
   RCDLABEL (CBX_KEK_0001)
   KEYTYPE (*AES)
   KEYSIZE (16)

4. CRTDTAKS DTAKS (QGPL/CBX192)
   KEKKS (QGPL/CBX192)
   KEKLABEL (CBX_KEK_0001)

5. CRTDTAK DTAKS (QGPL/CBX192)
   KEYLABEL (CBX_DTAK_0001)
   LENGTH (16)
   KEY1 (*GEN)
   KEY2 (*GEN)

```

When you've successfully completed the setup described above, the foundation and encryption key infrastructure necessary to encrypt and decrypt data will be in place on your system. The remaining part concerns the controlling and execution of the actual data encryption and decryption process. The first issue to address is the control data file, preserving the identity of the data encryption key, defined by key label and qualified key store name, to be used in the customer data encryption and decryption process. Because this sample application involves the creation of customer records, I've also included a *Last customer number* field to help me assign unique customer numbers as I create new customer records. Here's the outcome of my efforts:

```

                                Display File Field Description
WYNDHAMW
                                20-04-08
13:08:57
File . . . . . :   CBX1921F           Record length . :   60
Library . . . . :   QGPL               Field count . . :   4
Record format . :   CBX1921R
File type . . . :   PF
Access path . . :   *ARRIVAL

Field      Data type  Buffer  Length  Dig  Dec  Key  Text

```

KEYLBL	Char	1	32			Key label
KEYSTO	Char	33	10			Key store
KEYLIB	Char	43	10			Key store
library						
LSTCUS	Zoned	53	8	8	0	Last
customer number						

As you will note if you inspect the CBX192 service program included today, I've made the above information accessible through subprocedures. This approach enforces encapsulation of the file access, as well as simplifies the retrieval of the file data for the programs requiring the information stored in the file. The CBX192 service program also contains the subprocedures performing the retrieval and update services of the customer file records. Here's the simple customer file record layout:

Display File Field Description							
WYNHAMW							
20-04-08							
13:10:11							
File	:	CBX1922F		Record length . .	:	136	
Library	:	QGPL		Field count . . .	:	9	
Record format .	:	CBX1922R					
File type	:	PF					
Access path . . .	:	*KEYED					
Field	Data type	Buffer	Length	Dig	Dec	Key	Text
CUSNBR	Packed	1	5	8	0	1 A	Customer
number							
CUSNAM	Char	6	30				Customer
name							
CUSADR	Char	36	30				Customer
address							
CUSCTY	Char	66	20				Customer
address							
CUSZIP	Char	86	5				Zip code
CUSSTT	Char	91	2				State
CUSPHO	Char	93	12				Phone
CUSSSN	Char	105	16				Social
Security Number							
ENCRIV	Char	121	16				
Initialization vector							

The first seven fields in the CBX1922F customer file contain information that in this context has been deemed nonsensitive and therefore will be stored in the file in clear text. The customer's Social Security number (SSN) however, I do not want to be immediately accessible. The regular and intensive use of query products, interactive SQL, ODBC and JDBC integration to PC clients and other facilities providing easy database access and extraction options prompts early and careful consideration when planning database encryption requirements. And in this case, an SSN falls within the category of data to which you will probably want to control both access and manipulation.

An SSN by definition occupies 9 digits. Because I use the AES algorithm and an encryption block size of 16 bytes in this example, I need to define the SSN field size as 16 bytes, because the field size required to store the AES encrypted data basically must always be an even multiple of the AES encryption block size.

The 16-byte field size also leaves room for the padding performed by the AES encryption algorithm. Because of this padding, you must always reserve at least one byte within the encryption block size multiple to allow for padding to occur. If, for example, the data to be encrypted occupied the full 16-byte block size, AES would have added another full block of padding, resulting in a 32-byte-sized encryption output (cipher) string.

Some encryption algorithms as well as specific algorithm encryption modes enable an equally sized encryption input and output string, but this aspect of course needs to be taken carefully into account early on when deciding on encryption algorithms and designing databases.

Another issue to note relates to the definition of the database field to hold the encrypted data. The encryption algorithm is always applied to the binary value of the data that is processed, so the cryptographic process sees only the bits that it is operating on, not the signs and characters that appear to our eyes. That is actually in contrast to the methods and declarations that the database applies in order to preserve the appearance of characters and signs across character sets and encoding schemes.

To ensure that a cipher string is not made subject to conversion of any type, I need to tag such fields with a CCSID value that defines the field's content as a hexadecimal value. This is achieved by specifying the CCSID() keyword, with the special value 65535, indicating a hexadecimal content for the relevant fields in the CBX1922F DDS source below. Please note that the CCSID keyword is also valid for fields defined by SQL DDL statements:

```

A          R CBX1922R
**
A          CUSNBR          8P 0          COLHDG('Customer number')
A          CUSNAM          30A          COLHDG('Customer' 'name')
A          CUSADR          30A          COLHDG('Customer' 'address')
A          CUSCTY          20A          COLHDG('Customer' 'address')
A          CUSZIP          5A          COLHDG('Zip code')
A          CUSSTT          2A          COLHDG('State')
A          CUSPHO          12A          COLHDG('Phone')
A          CUSSSN          16A          COLHDG('Social' 'Security'
'Number')
A          CCSID(65535)
A          ENCRIV          16A          COLHDG('Initialization'
'vector')
A          CCSID(65535)

```

**

A

K CUSNBR

As you will note, the CCSID(65535) keyword is specified for both the *Social Security Number* field and the *Initialization vector* field. The latter contains the *Initial Chaining Value* (or salt) that is applied to the encryption process in order to avoid patterns emerging in the encrypted data across file records, so a salt is generated and stored in the *Initialization vector* field for each file record as it is created. The data in this field is then used when encrypting or decrypting the SSN. The salt does not require any confidentiality, but it needs to be protected against conversion of course, because if it is altered following the data encryption, the data decryption process will fail.

At this point, it is time to present the Add Customer Record (ADDCUSRCD) and Change Customer Record (CHGCUSRCD) commands, as they are providing the interface to the components and infrastructure discussed so far. The ADDCUSRCD command prompt displays the following parameters:

Add Customer Record (ADDCUSRCD)

Type choices, press Enter.

Customer name

Address

City

State

Zip code

Phone number

Social security number

All the customer data file fields require input, and there's a help text panel group providing a brief explanation of the command as well as each parameter. The ADDCUSRCD CPP uses the services provided by the CBX192 service program to encrypt and store the data in the CBX1922F data file. The customer number assigned to the customer record is returned in the completion message for future reference.

When the record is created, you can use the CHGCURRCD command to retrieve and alter the customer data. The CHGCUSRCD command prompt will initially display only the customer number field, but as soon as the operator enters the relevant customer number and presses Enter, all customer data is retrieved and displayed. Here's the full command prompt:

Change Customer Record (CHGCUSRCD)

Type choices, press Enter.

Customer number

Customer name

Address

City

State

Zip code

Phone number

Social security number

Again, online help text is provided. To control access to both commands, you'll need to register the user profiles requiring access to these commands through the function usage facility, and more specifically to the CBX_CRYPTO_KEY_USAGE special function.

The CBX192M CL program included with this article to create all commands and objects will automatically register the CBX_CRYPTO_KEY_USAGE special function usage and authorize the user profile running the CL program to this function. Adding or removing user profiles to the CBX_CRYPTO_KEY_USAGE special function will change these users' access to running the ADDCUSRCD and CHGCUSRCD commands accordingly. Here's how you can test that important part of the game:

Use DFU or some other data file utility to update the CBX1921F control file. Specify the data key label (in the above example, CBX_DTAK_0001) in the KEYLBL field as well as the key store name and library in the KEYSTO and KEYLIB fields, respectively. Specify whatever number you want to be the initial customer number in the field LSTCUS.

1. Run the command ADDCUSRCD to create a customer record.
2. Run the command RUNQRY *N CBX1922F to verify the record has been added and the SSN encrypted.
3. Run the command CHGCUSRCD, specifying the customer number returned in step 5. You should now be able to see in clear text the data previously entered. Try to change the data and repeat this step to verify the change.
4. Run the command WRKFCNUSG FCNID(CBX_CRYPTO_KEY_USAGE) and remove your function usage authorization: Option 2, specify USER() USAGE(*NONE). Now try to run step 4 again. Remember to reinstate your function usage authorization when your test is complete, if required.

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

```
WRKFCNUSG FCNID(CBX_CRYPTO_*)
```

Given that you've successfully loaded and installed the commands and usage registrations provided with this and the previous APIs by Example Cryptographic Key Management articles, you should see a list similar to the one below, following a successful execution of the command above:

```

                                Work with Function Usage

Type options, press Enter.

2=Change usage    5=Display usage


Opt  Function ID                                Function Name
    CBX_CRYPTO_KEY_MANAGEMENT                  Cryptographic key management
    CBX_CRYPTO_KEY_USAGE                      Cryptographic key management
    CBX_CRYPTO_KEYRECORD_DELETE              Cryptographic key record
deletion
    CBX_CRYPTO_KEYSTORE_XLATE                Cryptographic key store
translation
    CBX_CRYPTO_MASTERKEY_CLEAR              Clear cryptographic master key
load
    CBX_CRYPTO_MASTERKEY_LOAD              Cryptographic master key part
    CBX_CRYPTO_MASTERKEY_SET              Set cryptographic master key
    CBX_CRYPTO_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. As I mentioned, there's more information on function usage registration prerequisites and requirements in an earlier APIs by Example article, the one of December 13, 2007 (please follow the link provided below to look up that article).

This APIs by Example includes the following sources:

```
CBX191  -- RPGLE  -- Cryptographic Data Key Management - services
(update)
CBX191B -- SRVSRC -- Cryptographic Data Key Management - binder
source (update)

CBX192  -- RPGLE  -- Customer data - services
CBX192B -- SRVSRC -- Customer data - binder source

CBX1921F -- PF      -- Customer control data
CBX1922F -- PF      -- Customer data

CBX1921  -- RPGLE  -- Add Customer Record
CBX1921H -- PNLGRP -- Add Customer Record - Help
CBX1921V -- RPGLE  -- Add Customer Record - VCP
CBX1921X -- CMD     -- Add Customer Record

CBX1922  -- RPGLE  -- Change Customer Record
CBX1922H -- PNLGRP -- Change Customer Record - Help
CBX1922O -- RPGLE  -- Change Customer Record - POP
CBX1922V -- RPGLE  -- Change Customer Record - VCP
CBX1922X -- CMD     -- Change Customer Record

CBX192M  -- CLP     -- Cryptographic Data Key Management - build
commands
```

To create all these objects, compile and run CBX192M, following the instructions in the source header. As always, you'll also find compilation instructions in the respective source headers.

Please note that the two previously published commands Add Function Registration (ADDFCNREG) and Change User Function Usage (CHGUSRFCNU) are prerequisite for the CBX190M program to compile.

You can get the sources for the two aforementioned user function commands with the download made available with my previous APIs by Example article of November 8, 2007 -- just follow the link provided below. Successfully compiling and running the CBX180M CL setup program included with that article is also prerequisite to running the CBX192M setup program included today.

In a previous article series, I've presented similar commands for the very same purpose of demonstrating a practical cryptographic programming approach, so in case you've installed the previous versions of the ADDCUSRCD and CHGCUSRCD commands, the CBX192M setup program will rename the two commands to ADDCUSRCD2 and CHGCUSRCD2, respectively, to prevent them from being replaced by the newer versions.

Many of the issues and techniques described and demonstrated in this APIs by Example article and code have also been discussed in detail in previously published articles. I have included links to these articles below.

Before commencing any programming projects involving cryptographic requirements, I urge you to read and comprehend all the articles and also the warnings and recommendations stated therein. This precaution is especially important before using any of the code provided as part of the APIs by Example articles, as it is intended only as a starting point for your own efforts in this interesting but challenging programming discipline.

Previously published related articles:

Cryptographic Services APIs: Key Management:

<http://systeminetwork.com/article/cryptographic-services-apis-key-management>

APIs by Example, July 21, 2005: Cryptographic Services APIs, Part 1:

<http://systeminetwork.com/article/apis-example-cryptographic-services-apis>

APIs by Example, November 10, 2005: Cryptographic Services APIs, Part 2:

<http://systeminetwork.com/article/apis-example-cryptographic-services-apis-part-2>

APIs by Example, December 8, 2005: Cryptographic Services APIs, Part 3:

<http://systeminetwork.com/article/apis-example-cryptographic-services-apis-part-3>

APIs by Example, January 12, 2006: Cryptographic Services APIs, Part 4:

<http://systeminetwork.com/article/apis-example-cryptographic-services-apis-part-4>

APIs by Example, January 26, 2006: Cryptographic Services APIs, Part 5:

<http://systeminetwork.com/article/apis-example-cryptographic-services-apis-part-5>

APIs by Example, February 16, 2006: Cryptographic Services APIs, Part 6:

<http://systeminetwork.com/article/apis-example-cryptographic-services-apis-part-6>

APIs by Example, March 9, 2006: Cryptographic Services APIs, Part 7:

<http://systeminetwork.com/article/apis-example-cryptographic-services-apis-part-7>

APIs by Example, November 8, 2007: Cryptographic Key Management - Loading and Setting Master Keys::

<http://systeminetwork.com/article/apis-example-cryptographic-key-management-loading-and-setting-master-keys>

APIs by Example, December 13, 2007: Cryptographic Key Management - Testing and Clearing Master Keys:

<http://systeminetwork.com/article/apis-example-cryptographic-key-management-testing-and-clearing-master-keys>

APIs by Example, January 24, 2008: Cryptographic Key Management – Creating and Translating Key Stores:

<http://systeminetwork.com/article/apis-example-cryptographic-key-management-creating-and-translating-key-stores>

APIs by Example, February 28, 2008: Cryptographic Key Management – Creating, Displaying, and Deleting Key Records:

<http://systeminetwork.com/article/apis-example-cryptographic-key-management-%E2%80%93-creating-displaying-and-deleting-key-records>

APIs by Example, March 28, 2008: Cryptographic Key Management - Creating Data Key Stores and More:

<http://systeminetwork.com/article/apis-example-crypto-key-management-creating-data-key-stores-and-more>

IBM documentation:

Scenario: Key Management and File Encryption Using the Cryptographic Services APIs:

<http://publib.boulder.ibm.com/infocenter/iseriess/v5r4/topic/apis/qc3Scenario.htm>

Educational White Paper: Protecting i5/OS Data with Encryption:

http://www-03.ibm.com/servers/enablers/site/education/abstracts/efbe_abs.html

IBM System i Security: Protecting i5/OS Data with Encryption:

<http://www.redbooks.ibm.com/Redbooks.nsf/RedpieceAbstracts/sg247399.html?Open>

This article demonstrates the following Cryptographic Services API:

Create Key Store (Qc3CreateKeyStore) API:

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

Generate Key Record (Qc3GenKeyRecord) API:

<http://publib.boulder.ibm.com/infocenter/iseriess/v5r4/topic/apis/qc3genkr.htm>

Encrypt Data (Qc3EncryptData) API:

<http://publib.boulder.ibm.com/infocenter/iseriess/v5r4/topic/apis/qc3encdt.htm>

Decrypt Data (Qc3DecryptData) API:

<http://publib.boulder.ibm.com/infocenter/iseriess/v5r4/topic/apis/qc3decdt.htm>

Translate Data (Qc3TranslateData) API:

<http://publib.boulder.ibm.com/infocenter/iseriess/v5r4/topic/apis/qc3trndt.htm>

Generate Symmetric Key (Qc3GenSymmetricKey) API:

<http://publib.boulder.ibm.com/infocenter/iseriess/v5r4/topic/apis/qc3gensk.htm>

Generate Pseudorandom Numbers (Qc3GenPRNs) API:

<http://publib.boulder.ibm.com/infocenter/iseriess/v5r4/topic/apis/qc3genprns.htm>

Create Algorithm Context (Qc3CreateAlgorithmContext) API:

<http://publib.boulder.ibm.com/infocenter/iseriess/v5r4/topic/apis/qc3crtax.htm>

Create Key Context (Qc3CreateKeyContext) API:

<http://publib.boulder.ibm.com/infocenter/iseriess/v5r4/topic/apis/qc3crtkx.htm>

Destroy Algorithm Context (Qc3DestroyAlgorithmContext) API:

<http://publib.boulder.ibm.com/infocenter/iseriess/v5r4/topic/apis/qc3desax.htm>

Destroy Key Context (Qc3DestroyKeyContext) API:

<http://publib.boulder.ibm.com/infocenter/iseriess/v5r4/topic/apis/qc3deskx.htm>

Retrieve Key Record Attributes (Qc3RetrieveKeyRecordAttr) API:

<http://publib.boulder.ibm.com/infocenter/iseriess/v5r4/topic/apis/qc3rtvka.htm>

Key Management APIs V5R4:

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

Cryptographic Services APIs V5R4:

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

Validation List APIs V5R4:

<http://publib.boulder.ibm.com/infocenter/iserics/v5r4/topic/apis/sec6.htm>

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

http://www.pentontech.com/IBMContent/Documents/article/56586_572_KeyHierarchy.zip

Source URL: <http://iprodeveloper.com/rpg-programming/apis-example-crypto-key-mgmt-encryptdecrypt-key-hierarchy>