


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

APIs by Example: Cryptographic Services APIs, Part 6

[System iNetwork Programming Tips Newsletter](#)

[Carsten Flensburg](#)

Carsten Flensburg

Thu, 02/16/2006 (All day)

In the previous installments of this article series, I built a number of tools to establish an encryption key hierarchy: the Create Master Key (CRTMSTK) command, the Create Key Encrypting Key (CRTKEK) command, and the Create Data Encryption Key (CRTDTAK) command. In this installment, I put together all the pieces that I've provided so far and use these encryption key commands and a set of new functions to encrypt and decrypt sensitive data.

To perform this demonstration, I've written two sample commands, the Add Customer Record (ADDCUSRCD) and the Change Customer Record (CHGCUSRCD) command. The ADDCUSRCD command prompt has the following layout:

```

                                Add Customer Record (ADDCUSRCD)

Type choices, press Enter.

Customer name  . . . . . _____
Address       . . . . . _____
City         . . . . . _____
State        . . . . . _____
Zip code     . . . . . _____
Phone number  . . . . . _____
  
```

A successful execution of the ADDCUSRCD command returns a completion message specifying the assigned customer number:

```
Customer number 1 added.
```

The ADDCUSRCD CPP encrypts and stores the sensitive customer data in the CBX1502F data file.

Creating a customer record lets you also change that record. The CHGCUSRCD command initially looks like this:

```

                                Change Customer Record (CHGCUSRCD)

Type choices, press Enter.
Customer number  . . . . . _____
  
```

If you enter an existing customer number and have the required usage authorization (which I discuss momentarily), a prompt override program retrieves and decrypts the customer data for that customer. The customer data is then presented in the full command prompt:

```

                                Change Customer Record (CHGCUSRCD)
Type choices, press Enter.
Customer number . . . . . > 1
Customer name . . . . . 'George Best      '
Address . . . . . 'E. Washington St. 7 '
City . . . . . 'Phoenix                '
State . . . . . 'AZ'
Zip code . . . . . '85077'
Phone number . . . . . '6029328070'

```

Changing any of the customer data and pressing Enter causes all data to be encrypted and written to the customer record.

To run the preceding commands, a user must have usage authorization to the CBX_CRYPTO_KEY_USAGE user function, which was installed earlier during creation of command objects. The command WRKFCNUSG FCNID(CBX_CRYPTO_KEY_USAGE) shows you exactly which user profiles are authorized and also lets you add or remove user profiles.

Apart from this requirement, here's a quick setup guide to prepare the test:

1. Create a master key using the CRTMSTK command: CRTMSTK
2. Create a key-encrypting key using the CRTKEK command: CRTKEK KEYLABEL (CBX_KEK_0001)
3. Create a data encryption key using the CRTDTAK command: CRTDTAK KEYLABEL (CBX_DTAK_0001) KEKLABEL(CBX_KEK_0001)
4. Use Data File Utility (DFU) or some other such utility to update the CBX1501F control file. Specify the data key label (in the preceding example, CBX_DTAK_0001) in the KEYLBL field. Specify whatever number you want to be the initial customer number in the field LSTCUS.
5. Run the ADDCUSRCD command to create a customer record.
6. Run the RUNQRY *N CBX1502F command to verify that the record has been added and the customer data encrypted.
7. Run the CHGCUSRCD command, specifying the customer number returned in step 5. You should now be able to see in cleartext the data previously entered. Try to change the data and repeat this step to verify the change.
8. 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 7 again. (Remember to reinstall your function usage authorization when your test is complete, if required.)

The interfaces between the CPPs and the key store and cryptographic functions in the two previously published service programs, CBX146 and CBX147, are located in a new CBX150 service program, making the following functions available to the commands:

```

GetDtaAlg()  -- Get data encryption algorithm context
GetDtaKeyLb() -- Get customer data key label

```

```
GetNxtCusNo() -- Get next customer number
VfyCusDta()  -- Verify customer data
GetCusDta()  -- Get customer data
AddCusDta()  -- Add customer data
ChgCusDta()  -- Change customer data
```

During the research, development, and testing that I performed while writing these articles, I encountered a variety of concerns and topics related to the practical use of the Cryptographic Services APIs. One is related to the Advanced Encryption Algorithm's (AES's) use of padding, and it has an important impact on the space required to store the encrypted data.

As I've mentioned, the AES algorithm produces a cipher string length that is a multiple of the specified block length (which is 16 bytes in this article series). It does, however, also always apply padding to the cleartext string before encrypting it. This means that if you have a cleartext string length that is an exact multiple of the block length, AES adds a full block length of padding to the cleartext string.

In my example, the customer data field occupies 80 bytes in the customer file. This is because the actual customer data takes up 77 bytes, thus leaving 3 bytes for the padding. Having the Encrypt Data API encrypt the full 80-byte string, which is an exact multiple of 16, would cause the API to add another full block length of 16 bytes of pad characters, producing a cipher string of 96 bytes.

You must therefore pass only the 77 bytes of customer data to the Encrypt Data API, to ensure that it doesn't return a cipher string longer than the storage available in the file to hold it. I achieve this by trimming the blank pad field when passing the cleartext data to the Encrypt Data API.

A fairly comprehensive and detailed discussion of the concept of padding in cryptography is in "Using padding in Encryption":

<http://www.di-mgt.com.au/cryptopad.html>

Another cryptographic API implementation concern emerged while I was testing the commands that I present in this article. This concern was the context token behavior that requires the key and algorithm context tokens used in the process of generating a new key context token to remain valid in order for the new key context token to also be valid in a subsequent cryptographic process.

Let me explain that in a bit more detail: When I generate a data encryption key context token, I specify the key-encrypting key (KEK) context token and a key management algorithm context token as input to the Create Key Context API. This is necessary because, as you remember, this is how the data encryption key value is stored: Encrypted under a KEK. So to avoid having to decrypt the data encryption key before creating a data encryption key context, I simply pass the KEK context as well as the KEK algorithm context token to the Create Key Context API. This way the KEK itself is never exposed.

The result of this process is a data encryption key context token. This data encryption key context token, however, is deemed valid only by, for example the Encrypt Data API, if the key encryption key and algorithm context tokens used to produce the data encryption key context are also still valid (i.e., have not been destroyed or invalidated in the meantime).

So using key context tokens requires you to manage all involved context tokens for the full period of usage and perform the necessary housekeeping when they are no longer needed. Different methods are available to perform such housekeeping and should be carefully evaluated to minimize the risk of exposure in each individual application or context.

Reader Matt Haas brought yet another concern to my attention during the course of this article series. Matt pointed out the challenge of having to exchange encrypted data between different systems and different code pages and character sets. Because modern cryptographic methods always operate at the bit level, as opposed to the character level, it is of highest importance to take into consideration when and where data conversion is going to take place, if such a requirement is involved in your set up. Although it's beyond the scope of the cryptographic API example that I present here, you might want to read the very interesting and detailed discussion of this complex topic in "Encryption with International Character Sets":

<http://www.di-mgt.com.au/cryptoInternational.html>

And if you're hungry for even more cryptography reading, look up "An Overview of Cryptography," by Gary C. Kessler, which provides a comprehensive overview with many relevant links to even more detailed and specific information:

<http://www.garykessler.net/library/crypto.html>

As for the key management concern, please note that V5R4 has added a set of Key Management APIs. Check out the Cryptographic Services API manual for further details and new information about the subject:

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

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
GetDtaKeyTk() -- Get data key context token
RmvParCtxTk() -- Remove parent context tokens
```

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

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>

Part five here:

<http://www2.systeminetwork.com/article.cfm?id=52017>

This APIs by Example includes the following source members:

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

These sources are both revised versions of previously published sources, which have been updated to support the new functions that this article introduces. Please replace these sources in your utility library's source files. The CBX150M program ensures that the service program gets correctly recompiled.

The following new sources deliver the ADDCUSRCD and CHGCUSRCD commands and related services:

```
CBX150 -- Customer data management - service program
CBX150B -- Service program binder source

CBX1501 -- Add Customer Record - command processor
CBX1501H -- Add Customer Record - help
CBX1501V -- Add Customer Record - validity checker
CBX1501X -- Add Customer Record - command

CBX1502 -- Change Customer Record - command processor
CBX1502H -- Change Customer Record - help
CBX1502V -- Change Customer Record - validity checker
CBX1502O -- Change Customer Record - prompt override program
CBX1502X -- Change Customer Record - command
```

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

```
CBX150M -- 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 (QMHHMOVPM) 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/52119_58_CryptoServices6.zip

Source URL: <http://iprodeveloper.com/rpg-programming/apis-example-cryptographic-services-apis-part-6>