

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

APIs by Example: Cryptographic Services APIs, Part 3

[System iNetwork Programming Tips Newsletter](#)

[Carsten Flensburg](#)

Carsten Flensburg

Thu, 12/08/2005 (All day)

In today's issue of APIs by Example, I take up the challenge of cryptographic key management. How do you protect a cipher key against disclosure and at the same time make it readily available to your applications?

The idea behind key management is based on the simple fact that encrypted data is protected against unauthorized access only as long as the data encryption key is kept secret. Anyone having access to the data encryption key, and the encrypted data, of course, can get access to the clear-text data.

The simple answer to this problem is to encrypt the data encryption key. For that purpose, key-encrypting keys (KEKs) are implemented. So whenever a data encryption key is stored, it is first encrypted using a KEK.

This leaves you with the challenge of protecting the KEK against disclosure. So a final key layer is introduced: The master key. Using a master key, I can encrypt all KEKs before storing them on disk. And so they are also safe and protected. But now how do I protect the master key?

For the time being, the iSeries provides no built-in key management facilities. In her recent *iSeries NEWS* article, Beth Hagemeister of IBM says, "Additional help with key management is planned for the Cryptographic Services APIs in a future release." The whole article is available for *iSeries NEWS* ProVIP subscribers at the following link:

<http://www2.systeminetwork.com/Article.cfm?ID=20312>

Alas, for now, you have to invent your own key management facilities to be able to apply an appropriate level of protection to all key layers.

Being encrypted, the KEKs and the data encryption keys could be stored in a physical data file. Just remember to specify the CCSID(65535) keyword for the field holding the encryption key to prevent the possibility of the binary key value being converted when written to the file.

But a physical file is obviously not a safe place to store the unencrypted master key. Anyone with sufficient object and data authority to the file could use any number of iSeries native and third-party file editors and query utilities to immediately access the master key.

This inspired the idea to use validation lists (iSeries object type *VLDL) to store the master key. Validation lists by no means constitute an unbreakable key store, but from a risk-management perspective, they offer a better level of protection than data files, because of the following validation list properties:

- Validation list entries are accessible only through APIs
- The list entry key data is stored encrypted
- No Coded Character Set Identifier (CCSID) conversion occurs when storing a list entry

- System values control the audit of all list entry access

So after I have written the functions needed to implement validation lists as a key store, I can use the system audit facility to monitor all access to the key store validation list. The system audit journal entry contains the name of the program that accessed the validation list, so it's easy to scan for unauthorized program access. I can also look for unauthorized save and restore activity against the key store validation list.

I have further placed all key management functions in one service program and registered a specific Key Management User Function. Only user profiles that have explicitly (or through group profile) been granted usage authorization to this function can use the critical key management functions. This also prevents users who have *ALLOBJ special authority from using that special authority to gain access to the key management functions through the authorized interfaces.

For more information about user functions, please check out the information at the end of this article, where I offer links to the User Function API documentation at the iSeries Information center, as well as to a series of articles on this topic previously published in this APIs by Examples column.

The example key management facility that I start building in this article and complete in following installments is based on the three-level key management design:

1. Master Key
used to encrypt the Key Encryption Key(s)
2. Key Encryption Key(s)
used to encrypt the Data Encryption Key(s)
3. Data Encryption Key(s)
used to perform the actual data encryption

To simplify key administration procedures and functions, I store all three key types in the validation list key store.

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.

In this installment, I start off with the master key level and introduce the Create Master Key (CRTMSTK) and Remove Master Key (RMVMSTK) commands. Here's the prompt for CRTMSTK:

```

                                Create Master Key (CRTMSTK)
Type choices, press Enter.
Key length . . . . . 16                16, 24, 32
Key bytes 1-8 . . . . . *GEN
Key bytes 9-16 . . . . . *GEN
Key bytes 17-24 . . . . . *GEN
Key bytes 25-32 . . . . . *GEN

```

For the CRTMSTK command to run successfully, the system audit journal QAUDJRN must be installed, and the key store validation list must be owned by QSECOFR. Further, the system value

QRETSVRSEC (Retain Server Security) must be set on, and the QAUDLVL (Security Auditing Level) system value must be active and include either the *SECURITY or the *SECVLDDL option.

In upcoming installments, I will include commands and functions to also enable creating and removing key encryption keys as well as data encryption keys. I will further include commands that address the requirement of regularly changing the master key and key encryption keys.

Please note that this key management sample application is mainly intended to offer an introduction to the Cryptographic Services APIs and as a starting point anyone faced with the need to develop cryptographic applications. Careful research and design efforts are mandatory to ensure that any cryptographic application that you put into production is in accordance with your shop's specific requirements and development guidelines.

I've made the following cryptographic and key management functions available with this and previous articles:

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

As mentioned, 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>

This APIs by Example includes the following source members:

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

CBX1471H -- Create Master Key - help

CBX1471V -- Create Master Key - validity checker

CBX1471X -- Create Master key - command

CBX1472 -- Remove Master Key - command processing program

CBX1472H -- Remove Master Key - help

CBX1472V -- Remove Master Key - validity checker

CBX1472X -- Remove Master key - command

I have included a program that performs the necessary object-creation and system-setup activities.

Please review the source carefully before compiling and running the program to verify that all actions are acceptable to you and your shop:

CBX147M -- Object creation and setup program

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

User Function APIs:

<http://publib.boulder.ibm.com/infocenter/iserics/v5r3/topic/apis/sec4.htm>

APIs by Example User Function articles

- part one:

<http://www2.systeminetwork.com/article.cfm?ID=51361>

- part two:

<http://www2.systeminetwork.com/article.cfm?ID=51418>

- part three:

<http://www2.systeminetwork.com/article.cfm?ID=51525>

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/qmhmovpm.htm>

Resend Escape Message (QMHRSNEM) API:

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

Retrieve System Values (QWCRSVAL) API:

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

Retrieve Object Description (QUSROBJD) API:

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

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

http://www.pentontech.com/IBMContent/Documents/article/51863_48_CryptoServices3.zip

Source URL: <http://iprodeveloper.com/rpg-programming/apis-example-cryptographic-services-apis-part-3>