

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

APIs by Example: Cryptographic Services APIs, Part 2

[System iNetwork Programming Tips Newsletter](#)

[Carsten Flensburg](#)

Carsten Flensburg

Thu, 11/10/2005 (All day)

In V5R3, IBM has substantially enhanced the Cryptographic Services APIs, especially in the areas of key management and the securing of the cryptographic process. I begin uncovering these great improvements in today's issue, and in upcoming issues, I will provide you with the building blocks necessary to implement both a key management setup and the necessary cryptographic functions required to make such an infrastructure work.

The topic of today's article is the concept and use of context tokens in the cryptographic process. With V5R3, two types of context tokens were introduced: Algorithm context tokens and key context tokens.

As I pointed out in [the first installment of this series](#) (July 17, 2005, article ID 51236 at iSeriesNetwork.com), the cryptographic algorithms used herein are dependent on the presence of the IBM iSeries software product 5722-AC3 - Cryptographic Access Provider 128-bit for iSeries.

You can use the CL command DSPSFWRSC (Display Software Resources) to verify that this product is installed. If it's not installed, you can order it from IBM free of charge. If you're outside of the U.S., please note, however, that this product can be subject to U.S. export regulations.

An algorithm context token is a symbolic representation of the cryptographic algorithm and state to be used in the encryption and decryption process. Likewise, a key context token is a symbolic representation of a cryptographic key. The actual cryptographic algorithm and cipher key data is stored in a system repository and is accessible only through the context token returned when creating the context.

After it's created, the context token can be used as input to the cryptographic APIs instead of the actual algorithm settings and the cipher key, but only within the same job in which it was created. Because only the token is passed on to the cryptographic processes, the risk of exposing cryptographic algorithm and key values is potentially reduced.

In some situations, using context tokens improves the performance of the cryptographic process.

APIs are available to destroy context tokens explicitly; otherwise, that happens implicitly when the job that created them ends.

The test program that I provide with this article performs two full encryption and decryption processes. Again, I use the Advanced Encryption Standard (AES) block cipher algorithm, as I did in the first article.

The first pass repeats one of the tests from the CBX139T test program in the previous installment. But this time, I use algorithm and key context tokens instead of specifying the algorithm settings and the key value directly when calling the encryption and decryption APIs.

The second pass also uses context tokens and additionally demonstrates how to use an initialization vector, also known as a salt, in the cipher process. This is done by replacing the Electronic Code Book (ECB) mode used in the first pass with a Cipher Block Chaining (CBC) mode. Doing so lets me specify a salt that's up to 32 bytes long to initialize the cipher process. How many bytes will actually be used depends on the block size used in the cipher process. I use a block length of 16 in this example.

Using a salt in the cipher process is essential to avoid encrypting identical clear-text blocks to identical cipher-text blocks, which would seriously endanger message confidentiality.

Read the details about block cipher modes in this Wikipedia article titled "Block cipher modes of operation":

http://en.wikipedia.org/wiki/Block_cipher_modes_of_operation

At the end of the Wikipedia article is a brief reference to the concept of padding in the cipher process, and this information is worth having a closer look at. I have found different padding settings or implementations in cipher algorithms quite often to be the cause of problems in the process of cipher-text decryption or message authentication. This is the case especially when more than one party is involved in this process.

To run the test program, compile the service and test programs and issue the following command:

```
CALL CBX146T
```

When prompted by an inquiry message, enter a short string to encrypt, and press Enter. A window similar to this one is presented:

```
.....
:
:   Data string . . . : Batman & Robin
:   Key string . . . : 4D6F33EF1F98F8F7D4603AF71D4FE613
:   Algorithm context : 000100019B9C7B8C
:   Key context . . . : 000100007DE9263E
:
:                                     Bottom
:   F12=Cancel
:.....
```

You will see the string that you entered in the first line and the encryption key to be used in the cipher test in the second line. The encryption key is generated by the test program. Finally, an algorithm context token and a key context token are shown. Please note that the key and context values have been translated from their character form to hex nibbles by the test program, prior to displaying them.

Now press Enter again, and you will see the cipher string resulting from the encryption of the clear-text data string:

```
.....
:
:   Data string . . . : Batman & Robin
:   Algorithm context : 000100031E8A3B87
:   Key context . . . : 00010002E4ABCFC0
:   Cipher string . . : |v ||è|©Tîß)|J||
:.....
```

When you press Enter the next time, the decryption process is performed and the clear text recovered:

The second pass repeats the preceding steps, this time including the initialization vector in the cipher process. You can see this value in the third line in the following example:

Starting debug against the CBX146T program and stepping through the test examples using command key F10 is another recommended way to study the work of the Cryptographic Services APIs and the other functions included in the test program. To jump into the execution of the subprocedures, press F22 when you reach the subprocedure statements.

The following cryptographic functions are made available with this article:

- GenAesKey() - Generate AES cipher key
- GenInzVct() - Generate initialization vector
- GetAlgCtx() - Get algorithm context
- GetKeyCtx() - Get key context
- RmvAlgCtx() - Remove algorithm context
- RmvKeyCtx() - Remove key context
- EncDtaStr() - Encrypt data string using context tokens
- DecDtaStr() - Decrypt data string using context tokens

I will continue my 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 functions.

My objective with this series of articles is to provide you with the tools and functions necessary to implement a cryptographic scenario similar to the one described at the following link:

<http://publib.boulder.ibm.com/infocenter/iseriess/v5r3/topic/apis/qc3Scenario.htm>

You can find part one of this APIs by Example article here:

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

Please also note that the November issue of *iSeries NEWS* contains a very interesting article on the topic of the Cryptographic Services APIs, written by IBM's Beth Hagemeister:

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

This APIs by Example includes the following sources:

- CBX146 - Cryptographic services service program
- CBX146B - Service program binder source
- CBX146T - Test cryptographic services service program

Compilation instructions are in the source headers, as usual.

V5R3 Cryptographic Services APIs documentation:

<http://publib.boulder.ibm.com/infocenter/iseriess/v5r3/topic/apis/catcrypt.htm>

This article demonstrates the following APIs:

Encrypt data (Qc3EncryptData) API:

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

Decrypt data (Qc3DecryptData) API:

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

Generate Symmetric Key (Qc3GenSymmetricKey) API:

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

Generate Pseudorandom Numbers (Qc3GenPRNs) API:

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

Create Algorithm Context (Qc3CreateAlgorithmContext) API:

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

Create Key Context (Qc3CreateKeyContext) API:

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

Destroy Algorithm Context (Qc3DestroyAlgorithmContext) API:

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

Destroy Key Context (Qc3DestroyKeyContext) API:

<http://publib.boulder.ibm.com/infocenter/iseriess/v5r3/topic/apis/qc3deskx.htm>

Display Long Text (QUILNGTX) API:

<http://publib.boulder.ibm.com/infocenter/iseriess/v5r3/topic/apis/quilngtx.htm>

Send Program Message (QMHSNDPM) API:

<http://publib.boulder.ibm.com/infocenter/iseriess/v5r3/topic/apis/QMHSNDPM.htm>

Receive Program Message (QMHRVPM) API:

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

You can retrieve the source code for this API example at

http://www.pentontech.com/IBMContent/Documents/article/51786_45_CryptoServices2.zip.

Source URL: <http://iprodeveloper.com/rpg-programming/apis-example-cryptographic-services-apis-part-2>