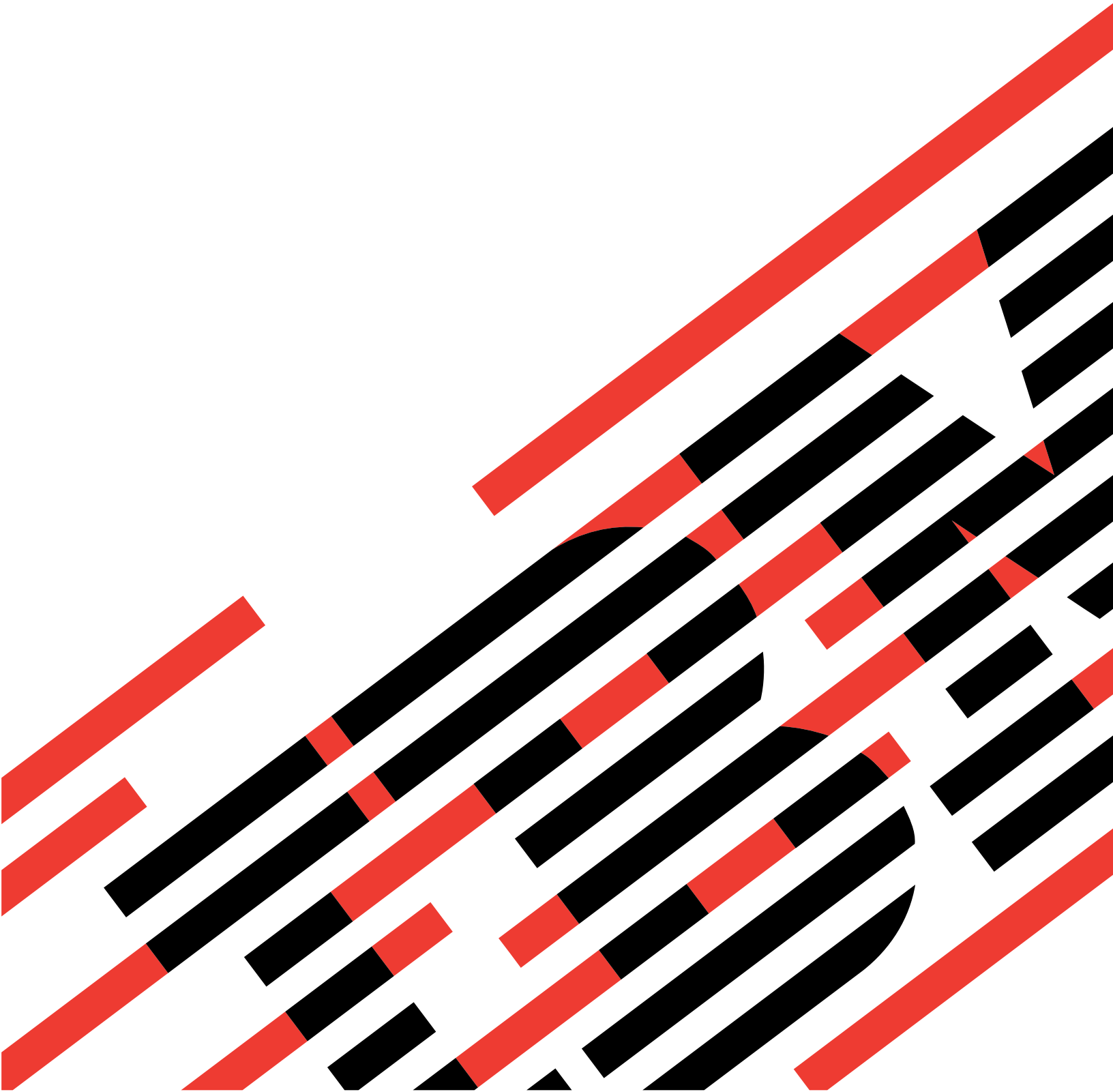
IBM

eserver

iSeries

Cryptographic Services APIs

Version 5 Release 3



IBM

@server

iSeries

Cryptographic Services APIs

Version 5 Release 3

Note

Before using this information and the product it supports, be sure to read the information in "Notices," on page 107.

Sixth Edition (August 2005)

This edition applies to version 5, release 3, modification 0 of Operating System/400 (product number 5722-SS1) and to all subsequent releases and modifications until otherwise indicated in new editions. This version does not run on all reduced instruction set computer (RISC) models nor does it run on CISC models.

Contents

Cryptographic Services APIs

The OS/400(R) Cryptographic Services APIs allow you to ensure the following:

- Privacy of data
- Integrity of data
- Authentication of communicating parties
- Non-repudiation of messages

For general information about cryptography, refer to Cryptographic Concepts in the Security topic.

The Cryptographic Services APIs perform cryptographic functions within the OS/400 or on the 2058 Cryptographic Accelerator for iSeries, as specified by the user. For more information about hardware cryptography, refer to Cryptographic Hardware in the Security topic. For a comparison of function performed in the OS/400 and on the 2058, refer to "OS/400 and 2058 Cryptographic Function Comparison" on page 83.

"Scenario: Key Management and File Encryption Using the Cryptographic Services APIs" on page 84 provides some sample designs and example programs.

Because cryptographic function is export controlled, the server is shipped with most cryptographic functions disabled. To enable full cryptographic capabilities, install 5722 AC3, Cryptographic Access Provider 128-bit for iSeries(TM).

The Cryptographic Services APIs include:

- "Encryption and Decryption APIs"
- "Authentication APIs" on page 22
- "Key Generation APIs" on page 53
- "Pseudorandom Number Generation APIs" on page 68
- "Cryptographic Context APIs" on page 72

APIs by category

APIs

These are the APIs for this category.

Encryption and Decryption APIs

The Encryption and Decryption APIs allow you to store information or to communicate with other parties while preventing uninvolved parties from understanding the stored information or understanding the communication. Encryption transforms understandable text (cleartext) into an unintelligible piece of data (ciphertext). Decrypting restores the cleartext from the ciphertext. Both processes involve a mathematical formula (algorithm) and secret data (key).

The Encryption and Decryption APIs include:

- "Decrypt Data (QC3DECDT, Qc3DecryptData)" on page 2 (QC3DECDT, Qc3DecryptData) restores encrypted data to a clear (intelligible) form.

- "Encrypt Data (QC3ENCDT, Qc3EncryptData)" on page 10 (QC3ENCDT, Qc3EncryptData) protects data privacy by scrambling clear data into an unintelligible form.
- "Translate Data (QC3TRNDT, Qc3TranslateData)" on page 19 (QC3TRNDT, Qc3TranslateData) translates data from encryption under one key to encryption under another key

Top | Cryptographic Services APIs | APIs by category

Decrypt Data (QC3DECDT, Qc3DecryptData)

Required Parameter Group:

1	Encrypted data
Input	Char(*)
2	Length of encrypted data
Input	Binary(4)
3	Algorithm description
Input	Char(*)
4	Algorithm description format name
Input	Char(8)
5	Key description
Input	Char(*)
6	Key description format name
Input	Char(8)
7	Cryptographic service provider
Input	Char(1)
8	Cryptographic device name
Input	Char(10)
9	Clear data
Output	Char(*)
10	Length of area provided for clear data
Input	Binary(4)
11	Length of clear data returned
Output	Binary(4)
12	Error code
I/O	Char(*)

Service Program Name: QC3DTADE

Default Public Authority: *USE

Threadsafe: Yes

The **Decrypt Data (OPM, QC3DECDT; ILE, Qc3DecryptData) API** restores encrypted data to a clear (intelligible) form.

Information on cryptographic standards can be found in the “Create Algorithm Context (QC3CRTAX, Qc3CreateAlgorithmContext)” on page 72 documentation.

Authorities and Locks

Required API authority
*USE

Required device description authority
*USE

Required Parameter Group

Encrypted data
INPUT; CHAR(*)

The data to decrypt.

Length of encrypted data
INPUT; BINARY(4)

The length of the encrypted data parameter.
If the mode of operation is CFB 1-bit, this length must be specified in bits.

Algorithm description
INPUT; CHAR(*)

The algorithm and associated parameters for decrypting the data.
The format of the algorithm description is specified in the algorithm description format name parameter.

Algorithm description format name
INPUT; CHAR(8)

The format of the algorithm description.
The possible format names follow.

“ALGD0100 format” on page 5
The token for an algorithm context. This format must be used when performing the decrypt operation over multiple calls. After the last call (when the final operation flag is on), the context will reset to its initial state and can be used in another API.

“ALGD0200 format” on page 5
Parameters for a block cipher algorithm (DES, Triple DES, AES, and RC2).

“ALGD0300 format” on page 5
Parameters for a stream cipher algorithm (RC4-compatible).

“ALGD0400 format” on page 5
Parameters for a public key algorithm (RSA).

See “Algorithm Description Formats” on page 5 for a description of these formats.

Key description
INPUT; CHAR(*)

The key and associated parameters for decrypting the data.
The format of the key description is specified in the key description format name parameter.
If the decrypt operation extends over multiple calls (see ALGD0100 description above), only the key description from the first call will be used. Therefore, on subsequent calls, you may set the pointer to this parameter to NULL.

Key description format name
INPUT; CHAR(8)

The format of the key description.
If the pointer to the key description parameter is NULL, this parameter will be ignored.
The possible format names follow.

"KEYD0100 format" on page 7
The token for a key context. This format must be used if the key is encrypted.

"KEYD0200 format" on page 7
Key parameters.

See "Key Description Formats" on page 7 for a description of these formats.

Cryptographic service provider
INPUT; CHAR(1)

The cryptographic service provider (CSP) that will perform the decryption operation.

0 Any CSP.
The system will choose an appropriate CSP to perform the decryption operation.

1 Software CSP.
The system will perform the decryption operation using software. If the requested algorithm is not available in software, an error is returned.

2 Hardware CSP.
The system will perform the decryption operation using cryptographic hardware. If the requested algorithm is not available in hardware, an error is returned. A specific cryptographic device can be specified using the cryptographic device name parameter. If the cryptographic device is not specified, the system will choose an appropriate one.

Cryptographic device name
INPUT; CHAR(10)

The name of a cryptographic device description.
This parameter is valid when the cryptographic service provider parameter specifies 2 (hardware CSP). Otherwise, this parameter must be blanks or the pointer to this parameter set to NULL.

Clear data
OUTPUT; CHAR(*)

The area to store the decrypted data.

Length of area provided for clear data
INPUT; BINARY(4)

The length of the clear data parameter.
If the mode of operation is CFB 1-bit, this length must be specified in bits.
To ensure sufficient space, specify an area at least as large as the length of encrypted data. If the length of area provided for clear data is too small, an error will be generated and no data will be returned in the clear data parameter.

Length of clear data returned
OUTPUT; BINARY(4)

The length of the clear data returned in the clear data parameter.
If the mode of operation is CFB 1-bit, this length will be returned in bits.

Error code
I/O; CHAR(*)

The structure in which to return error information.
For the format of the structure, see Error Code Parameter.

Algorithm Description Formats

For detailed descriptions of the table fields, see "**Algorithm Description Formats Field Descriptions**."

ALGD0100 format

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	CHAR(8)	Algorithm context token
8	8	CHAR(1)	Final operation flag

ALGD0200 format

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	BINARY(4)	Block cipher algorithm
4	4	BINARY(4)	Block length
8	8	CHAR(1)	Mode
9	9	CHAR(1)	Pad option
10	A	CHAR(1)	Pad character
11	B	CHAR(1)	Reserved
12	C	BINARY(4)	MAC length
16	10	BINARY(4)	Effective key size
20	14	CHAR(32)	Initialization vector

ALGD0300 format

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	BINARY(4)	Stream cipher algorithm

ALGD0400 format

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	BINARY(4)	Public key cipher algorithm
4	4	CHAR(1)	PKA block format
5	5	CHAR(3)	Reserved
8	8	BINARY(4)	Signing hash algorithm

Algorithm Description Formats Field Descriptions

Algorithm context token

A token for an algorithm context. The algorithm context is created using the "Create Algorithm Context (QC3CRTAX, Qc3CreateAlgorithmContext)" on page 72.

Block cipher algorithm

The decryption algorithm. Following are the valid block cipher algorithms.

20 DES
21 Triple DES

22 AES
23 RC2

Block length
The algorithm block length. For DES, Triple DES, and RC2, the block length field must specify 8. The valid block length values for AES are 16, 24, and 32. Note: 16 is the only block length specified in the AES standard, FIPS-197.

Effective key size
For RC2, the number of key bits to use in the cipher operation. Valid values are from 1 to 1024. If RC2 is not specifed for the block cipher algorithm, this field must be set to 0.

Final operation flag
The final processing indicator.

0 Continue.
The system will not perform final processing and the algorithm context will maintain the state of the operation. The algorithm context can be used on future calls to this API to continue the decryption operation.
1 Final.
The system will perform final processing (e.g. remove padding) and the algorithm context will reset to its initial state. The algorithm context can then be used to begin a new cryptographic operation (encrypt, decrypt, etc.). When performing a final operation, the pointer to the encrypted data parameter may be set to NULL and the length of encrypted data parameter set to 0. Final must be specified when performing an RSA operation.

Initialization vector
The initialization vector (IV). An IV is not used for mode ECB, and must be set to nulll (binary 0's). Refer to the mode standards for an explanation of its use. For DES, Triple DES, and RC2, the first 8 bytes are used as the IV. For AES, the length of IV used is that specified by block length. The IV must be the same as the IV used to encrypt the data.

MAC length
This field is not used on a decrypt operation and must be set to null (binary 0s).

Mode The mode of operation. Information on modes can be found in FIPS PUB 81 and ANSI X9.52. Following are the valid modes.

0 ECB
1 CBC
2 OFB. Not valid with AES or RC2.
3 CFB 1-bit. Not valid with AES or RC2.
4 CFB 8-bit. Not valid with AES or RC2.
5 CFB 64-bit. Not valid with AES or RC2.

Pad character
This field is not used on a decrypt operation and must be set to null (binary 0s).

Pad option
If requested, padding is removed at the end of the decrypt operation. Padding is not performed for modes CFB 1-bit and CFB 8-bit. In these cases, the pad option must be set to 0. Do not specify remove padding if the data was not padded when encrypted. Following are the valid pad options.

0 Do not remove padding.
1 Remove padding.

PKA block format
The public key algorithm block format. Following are the valid values.

0 PKCS #1 block type 00
1 PKCS #1 block type 01
2 PKCS #1 block type 02
This format is recommended when decrypting non-hash items (such as keys). The other formats are normally used in sign and verify functions.
4 Zero pad
Zero pad is not removed.

Public key cipher algorithm
The decryption algorithm. Following are the valid public key cipher algorithms.

50 RSA

Reserved
Must be null (binary 0s).

Signing hash algorithm
This field is not used on a decrypt operation and must be set to null (binary 0s).

Stream cipher algorithm
The decryption algorithm. Following are the valid stream cipher algorithms.

30 RC4-compatible

Key Description Formats

For detailed descriptions of the table fields, see "**Key Description Formats Field Descriptions**" on page 8.

KEYD0100 format

Offset		Type	Field
Dec	**Hex**		
0	0	CHAR(8)	Key context token

KEYD0200 format

Offset		Type	Field
Dec	**Hex**		
0	0	BINARY(4)	Key type
4	4	BINARY(4)	Key string length
8	8	CHAR(1)	Key format
9	9	CHAR(3)	Reserved
12	C	CHAR(*)	Key string

Key Description Formats Field Descriptions

Key context token

A token for a key context. The key context is created using the "Create Key Context (QC3CRTKX, Qc3CreateKeyContext)" on page 78.

Key format

The format of the key string field. Following are the valid values.

0 Binary string.
The key is specified as a binary value.

1 BER string
If the key type field specifies 50 (RSA public), the key must be specified in BER encoded X.509 SubjectPublicKeyInfo format. For specifications of this format, refer to RFC 3280. If the key type field specifies 51 (RSA private), the key must be specified in BER encoded PKCS #8 format. For specifications of this format, refer to RSA Security Inc. Public-Key Cryptography Standards. To generate a PKA key pair, use the "Generate PKA Key Pair (QC3GENPK, Qc3GenPKAKeyPair)" on page 61.

Key string

The key to use in the decrypt operation.

Key string length

Length of the key string specified in the key string field.

Key type

The type of key. Following are the valid values.

20 DES
The key format must be 0. The key string must be 8 bytes in length. Only 7 bits of each byte are used as the actual key. The rightmost bit of each byte is used to set parity. Some cryptographic service providers require that a DES key have odd parity in every byte. Others ignore parity.

21 Triple DES
The key format must be 0. The key string can be 8, 16, or 24 bytes in length. Triple DES operates on a decryption block by doing a DES decrypt, followed by a DES encrypt, and then another DES decrypt. Therefore, it actually uses three 8-byte DES keys. If 24 bytes are supplied in the key string, the first 8 bytes are used for key 1, the second 8 bytes for key 2, and the third 8 bytes for key 3. If 16 bytes are supplied, the first 8 bytes are used for key 1 and key 3, and the second 8 bytes for key 2. If only 8 bytes are supplied, it will be used for all 3 keys (essentially making the operation equivalent to a single DES operation). Only 7 bits of each byte are used as the actual key. The rightmost bit of each byte is used to set parity. Some cryptographic service providers require that a Triple DES key have odd parity in every byte. Others ignore parity.

22 AES
The key format must be 0. The key string can be 16, 24, or 32 bytes in length.

23 RC2
The key format must be 0. The key string can be from 1 to 128 bytes in length.

30 RC4-compatible
The key format must be 0. The key string can be from 1 to 256 bytes in length.

50 RSA public
The key format must be 1. Use an RSA public key if the data was encrypted with an RSA private key. Encryption with a private key and decryption with a public key is used for data authentication (e.g. sign/verify).

51 RSA private
The key format must be 1. Use an RSA private key if the data was encrypted with an RSA public key. Encryption with a public key and decryption with a private key is used for data privacy.

Reserved

Must be null (binary 0s).

Error Messages

Message ID	Error Message Text
CPF24B4 E	Severe error while addressing parameter list.
CPF3C1E E	Required parameter &1 omitted.
CPF3CF1 E	Error code parameter not valid.
CPF3CF2 E	Error(s) occurred during running of &1 API.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.
CPF9DC2 E	Key-encrypting algorithm context not compatible with key-encrypting key context.

Message ID	Error Message Text
CPF9DC3 E	Encrypted data contains invalid padding.
CPF9DC6 E	Algorithm not valid for encrypting or decrypting a key.
CPF9DC8 E	The input data parameter specifies a NULL pointer.
CPF9DD2 E	Algorithm description format name not valid.
CPF9DD3 E	Key description format name not valid.
CPF9DD5 E	Length of input data not valid.
CPF9DD6 E	Length of area provided for output data is too small.
CPF9DD7 E	The key-encrypting key context for the specified key is not valid or was previously destroyed.
CPF9DD8 E	The key-encrypting algorithm context for the specified key is not valid or was previously destroyed.
CPF9DD9 E	Effective key size not valid.
CPF9DDA E	Unexpected return code &1.
CPF9DDB E	The key string or Diffie-Hellman parameter string is not valid.
CPF9DDD E	The key string length is not valid.
CPF9DDE E	Cipher algorithm not valid.
CPF9DDF E	Block length not valid.
CPF9DE0 E	Hash algorithm not valid.
CPF9DE1 E	Initialization vector not valid.
CPF9DE2 E	MAC (message authentication code) length not valid.
CPF9DE3 E	Mode not valid.
CPF9DE4 E	Pad option not valid.
CPF9DE5 E	PKA (public key algorithm) block format not valid.
CPF9DE6 E	Public key algorithm not valid.
CPF9DE7 E	Key type not valid.
CPF9DE9 E	Key format not valid.
CPF9DEC E	Cryptographic service provider not valid.
CPF9DED E	Final operation flag not valid.
CPF9DEE E	Reserved field not null.
CPF9DF0 E	Operation, algorithm, or mode not available on the requested CSP (cryptographic service provider).
CPF9DF1 E	The algorithm context token does not reference a valid algorithm context.
CPF9DF2 E	The algorithm context is not found or was previously destroyed.
CPF9DF3 E	Algorithm in algorithm context not valid for requested operation.
CPF9DF4 E	The key context token does not reference a valid key context.
CPF9DF5 E	The key context is not found or was previously destroyed.
CPF9DF7 E	Algorithm context not compatible with key context.
CPF9DF8 E	Cryptographic device name not valid.
CPF9DF9 E	Cryptographic device not found.
CPF9DFB E	Cryptographic service provider (CSP) conflicts with the key context CSP.
CPF9DFD E	Not authorized to device.
CPF9DFE E	Cryptographic device not available.
CPF9DFF E	Request not allowed by cryptographic attributes.

API introduced: V5R3

Top | “Cryptographic Services APIs,” on page 1 | APIs by category

Encrypt Data (QC3ENCDT, Qc3EncryptData)

Required Parameter Group:

1	Clear data
Input	Char(*)
2	Length of clear data
Input	Binary(4)
3	Clear data format name
Input	Char(8)
4	Algorithm description
Input	Char(*)
5	Algorithm description format name
Input	Char(8)
6	Key description
Input	Char(*)
7	Key description format name
Input	Char(8)
8	Cryptographic service provider
Input	Char(1)
9	Cryptographic device name
Input	Char(10)
10	Encrypted data
Output	Char(*)
11	Length of area provided for encrypted data
Input	Binary(4)
12	Length of encrypted data returned
Output	Binary(4)
13	Error code
I/O	Char(*)

Service Program Name: QC3DTAEN

Default Public Authority: *USE

Threadsafe: Yes

The **Encrypt Data (OPM, QC3ENCDT; ILE, Qc3EncryptData) API** protects data privacy by scrambling clear data into an unintelligible form. To recover the clear data from the encrypted data, use the "Decrypt Data (QC3DECDT, Qc3DecryptData)" on page 2.

Information on cryptographic standards can be found in the "Create Algorithm Context (QC3CRTAX, Qc3CreateAlgorithmContext)" on page 72 documentation.

Authorities and Locks

Required API authority
*USE

Required device description authority
*USE

Required Parameter Group

Clear data
INPUT; CHAR(*)

The data to encrypt.
The format of the clear data is specified in the clear data format name parameter

Length of clear data
INPUT; BINARY(4)

For clear data format DATA0100, this is the length of the data to encrypt. For restrictions on the length of clear data, refer to the clear data length field below.
For clear data format DATA0200, this is the number of entries in the array.

Clear data format name
INPUT; CHAR(8)

The format of the clear data parameter.
The possible format names follow.

DATA0100
The clear data parameter contains the data to encrypt.

"DATA0200 format" on page 13
The clear data parameter contains an array of pointers and lengths to the data to encrypt. See "Clear Data Formats" on page 13 for a description of this format.

Algorithm description
INPUT; CHAR(*)

The algorithm and associated parameters for encrypting the data.
The format of the algorithm description is specified in the algorithm description format name parameter.

Algorithm description format name
INPUT; CHAR(8)

The format of the algorithm description.
The possible format names follow.

"ALGD0100 format" on page 14
The token for an algorithm context. This format must be used when performing the encrypt operation over multiple calls. After the last call (when the final operation flag is on), the context will reset to its initial state and can be used in another API.

"ALGD0200 format" on page 14
Parameters for a block cipher algorithm (DES, Triple DES, AES, and RC2).

"ALGD0300 format" on page 14
Parameters for a stream cipher algorithm (RC4-compatible).

"ALGD0400 format" on page 14
Parameters for a public key algorithm (RSA).

See “Algorithm Description Formats” on page 13 for a description of these formats.

Key description
INPUT; CHAR(*)

The key and associated parameters for encrypting the data.
The format of the key description is specified in the key description format name parameter.
If the encrypt operation extends over multiple calls (see ALGD0100 description above), only the key description from the first call will be used. Therefore, on subsequent calls, you may set the pointer to this parameter to NULL.

Key description format name
INPUT; CHAR(8)

The format of the key description.
If the pointer to the key description parameter is NULL, this parameter will be ignored.
The possible format names follow.

“KEYD0100 format” on page 16
The token for a key context. This format must be used if the key is encrypted.

“KEYD0200 format” on page 16
Key parameters.

See “Key Description Formats” on page 16 for a description of these formats.

Cryptographic service provider
INPUT; CHAR(1)

The cryptographic service provider (CSP) that will perform the encryption operation.

0 Any CSP.
The system will choose an appropriate CSP to perform the encryption operation.

1 Software CSP.
The system will perform the encryption operation using software. If the requested algorithm is not available in software, an error is returned.

2 Hardware CSP.
The system will perform the encryption operation using cryptographic hardware. If the requested algorithm is not available in hardware, an error is returned. A specific cryptographic device can be specified using the cryptographic device name parameter. If the cryptographic device is not specified, the system will choose an appropriate one.

Cryptographic device name
INPUT; CHAR(10)

The name of a cryptographic device description.
This parameter is valid when the cryptographic service provider parameter specifies 2 (hardware CSP). Otherwise, this parameter must be blanks or the pointer to this parameter set to NULL.

Encrypted data
OUTPUT; CHAR(*)

The area to store the encrypted data.

Length of area provided for encrypted data
INPUT; BINARY(4)

The length of the encrypted data parameter.
If the mode of operation is CFB 1-bit, this length must be specified in bits.
If the length of area provided for encrypted data is too small, an error will be generated and no data will be returned in the encrypted data parameter.

Block ciphers	The encrypted data parameter must be greater than or equal to the length of clear data. If padding and performing final processing, the encrypted data parameter must be large enough to include the pad characters. For more information, refer to the pad option description.
Stream ciphers	The encrypted data parameter must be greater than or equal to the length of clear data.
PKA ciphers	The encrypted data parameter must be greater than or equal to the key size.

Length of encrypted data returned
OUTPUT; BINARY(4)

The length of encrypted data returned in the encrypted data parameter.
If the mode of operation is CFB 1-bit, this length will be returned in bits.

Error code
I/O; CHAR(*)

The structure in which to return error information.
For the format of the structure, see Error Code Parameter.

Clear Data Formats

For detailed descriptions of the table fields, see "**Clear Data Formats Field Descriptions**."

DATA0200 format

Offset			
Dec	**Hex**	**Type**	**Field**
These fields repeat.		PTR(SPP)	Clear data pointer
		BINARY(4)	Clear data length
		CHAR(12)	Reserved

Clear Data Formats Field Descriptions

Clear data length
The length of data to encrypt. Some cipher algorithms have restrictions on the clear data length.

DES, Triple DES, AES, RC2
When mode is 0 (ECB), 2 (OFB), or 5 (CFB 64-bit) and pad option is 0 (no pad), the total of the clear data lengths for the entire encrypt operation must be a multiple of the block length. For mode 3 (CFB 1-bit), the clear data length is specified in bits, not bytes.

RSA For PKA block formats 0, 1, and 2, the clear data length must be at least 11 bytes shorter than the key size. For PKA block format 4, the data to encrypt must be shorter than or equal to the key size.

Clear data pointer
A space pointer to the data to encrypt.

Reserved
Must be null (binary 0s).

Algorithm Description Formats

For detailed descriptions of the table fields, see "**Algorithm Description Formats Field Descriptions**" on page 14.

ALGD0100 format

Offset		Type	Field
Dec	Hex		
0	0	CHAR(8)	Algorithm context token
8	8	CHAR(1)	Final operation flag

ALGD0200 format

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Block cipher algorithm
4	4	BINARY(4)	Block length
8	8	CHAR(1)	Mode
9	9	CHAR(1)	Pad option
10	A	CHAR(1)	Pad character
11	B	CHAR(1)	Reserved
12	C	BINARY(4)	MAC length
16	10	BINARY(4)	Effective key size
20	14	CHAR(32)	Initialization vector

ALGD0300 format

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Stream cipher algorithm

ALGD0400 format

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Public key cipher algorithm
4	4	CHAR(1)	PKA block format
5	5	CHAR(3)	Reserved
8	8	BINARY(4)	Signing hash algorithm

Algorithm Description Formats Field Descriptions

Algorithm context token
A token for an algorithm context. The algorithm context is created using the "Create Algorithm Context (QC3CRTAX, Qc3CreateAlgorithmContext)" on page 72.

Block cipher algorithm
The encryption algorithm. Following are the valid block cipher algorithms.

20 DES
21 Triple DES
22 AES
23 RC2

Block length
The algorithm block length. For DES, Triple DES, and RC2 the block length field must specify 8. The valid block length values for AES are 16, 24, and 32. Note: 16 is the only key size used by the AES standard, FIPS-197. Note: 16 is the only block length specified in the AES standard, FIPS-197.

Effective key size
For RC2, the number of key bits to use in the cipher operation. Valid values are from 1 to 1024. If RC2 is not specifed for the block cipher algorithm, this field must be set to 0.

Final operation flag
The final processing indicator.

0 Continue.
The system will not perform final processing and the algorithm context will maintain the state of the operation. The algorithm context can be used on future calls to this API to continue the encryption operation.

1 Final.
The system will perform final processing (e.g. padding) and the algorithm context will reset to its initial state. The algorithm context can then be used to begin a new cryptographic operation (encrypt, decrypt, etc.). When performing a final operation, the pointer to the clear data parameter may be set to NULL and the length of clear data parameter set to 0. This option must be specified if performing RSA encryption.

Initialization vector
The initialization vector (IV). An IV is not used for mode ECB, and must be set to NULL (binary 0s). Refer to the mode standards for an explanation of its use. For DES, Triple DES, and RC2, the first 8 bytes are used as the IV. For AES, the length of IV used is that specified by block length. The IV need not be secret, but it should be unique for each message. If not unique, it may compromise security. The IV can be any value. To obtain a good random IV value, use the "Generate Pseudorandom Numbers (QC3GENRN,Qc3GenPRNs) API" on page 70.

MAC length
This field is not used on an encrypt operation and must be set to null (binary 0s).

Mode The mode of operation. Information on modes can be found in FIPS PUB 81 and ANSI X9.52. Following are the valid modes.

0 ECB
1 CBC
2 OFB. Not valid with AES or RC2.
3 CFB 1-bit. Not valid with AES or RC2.
4 CFB 8-bit. Not valid with AES or RC2.
5 CFB 64-bit. Not valid with AES or RC2.

Pad character
The pad character for pad option 1. Using hex 00 as the pad character is equivalent to ANSI X9.23 padding.

Pad option
If requested, padding is performed at the end of the encrypt operation. Be sure the encrypted data parameter is large enough to include any padding. The data will be padded up to the next block length byte multiple. For example, when using DES and total data to encrypt is 20, the text is padded to 24. The last byte is filled with a 1-byte binary counter containing the number of pad characters used. The preceeding pad characters are filled as specified by this field. Padding is not performed for modes CFB 1-bit and CFB 8-bit. In these cases, the pad option must be set to 0. Following are the valid pad options.

0 No padding is performed.
1 Use the character specified in the pad character field for padding.
2 The pad counter is used as the pad character. This is equivalent to PKCS #5 padding.

PKA block format
The public key algorithm block format. Following are the valid values.

0 PKCS #1 block type 00
1 PKCS #1 block type 01
2 PKCS #1 block type 02
This format is recommended when encrypting non-hash items (such as keys). The other formats are normally used in sign and verify functions.
4 Zero pad
The clear data is placed in the low-order bit positions of a string of the same bit-length as the key modulus. All leading bits are set to zero.

Public key cipher algorithm
The encryption algorithm. Following are the valid public key cipher algorithms.

50 RSA

Reserved
Must be null (binary 0s).

Signing hash algorithm
This fieldis not used on an encrypt operation and must be set to null (binary 0s).

Stream cipher algorithm
The encryption algorithm. Following are the valid stream cipher algorithms.

30 RC4-compatible

Key Description Formats

For detailed descriptions of the table fields, see "**Key Description Formats Field Descriptions**."

KEYD0100 format

Offset		Type	Field
Dec	**Hex**		
0	0	CHAR(8)	Key context token

KEYD0200 format

Offset		Type	Field
Dec	**Hex**		
0	0	BINARY(4)	Key type
4	4	BINARY(4)	Key string length
8	8	CHAR(1)	Key format
9	9	CHAR(3)	Reserved
12	C	CHAR(*)	Key string

Key Description Formats Field Descriptions

Key context token
A token for a key context. The key context is created using the "Create Key Context (QC3CRTKX, Qc3CreateKeyContext)" on page 78.

Key format
The format of the key string field. Following are the valid values.

0 Binary string.
The key is specified as a binary value. To obtain a good random key value, use the "Generate Symmetric Key (QC3GENSK, Qc3GenSymmetricKey)" on page 65, or "Generate Pseudorandom Numbers (QC3GENRN,Qc3GenPRNs) API" on page 70 API.

1 BER string
If the key type field specifies 50 (RSA public), the key must be specified in BER encoded X.509 SubjectPublicKeyInfo format. For specifications of this format, refer to RFC 3280. If the key type field specifies 51 (RSA private), the key must be specified in BER encoded PKCS #8 format. For specifications of this format, refer to RSA Security Inc. Public-Key Cryptography Standards. To generate a PKA key pair, use the "Generate PKA Key Pair (QC3GENPK, Qc3GenPKAKeyPair)" on page 61.

Key string
The key to use in the encrypt operation.

Key string length
Length of the key string specified in the key string field.

Key type
The type of key. Following are the valid values.

20 DES
The key format must be 0. The key string must be 8 bytes in length. Only 7 bits of each byte are used as the actual key. The rightmost bit of each byte is used to set parity. Some cryptographic service providers require that a DES key have odd parity in every byte. Others ignore parity.

21 Triple DES
The key format must be 0. The key string can be 8, 16, or 24 bytes in length. Triple DES operates on an encryption block by doing a DES encrypt, followed by a DES decrypt, and then another DES encrypt. Therefore, it actually uses three 8-byte DES keys. If 24 bytes are supplied in the key string, the first 8 bytes are used for key 1, the second 8 bytes for key 2, and the third 8 bytes for key 3. If 16 bytes are supplied, the first 8 bytes are used for key 1 and key 3, and the second 8 bytes for key 2. If only 8 bytes are supplied, it will be used for all 3 keys (essentially making the operation equivalent to a single DES operation). Only 7 bits of each byte are used as the actual key. The rightmost bit of each byte is used to set parity. Some cryptographic service providers require that a Triple DES key have odd parity in every byte. Others ignore parity.

22 AES
The key format must be 0. The key string can be 16, 24, or 32 bytes in length.

23 RC2
The key format must be 0. The key string can be from 1 to 128 bytes in length.

30 RC4-compatible
The key format must be 0. The key string can be from 1 to 256 bytes in length. Because of the nature of the RC4-compatible operation, using the same key for more than one message will severely compromise security.

50 RSA public
The key format must be 1. Encryption with a public key and decryption with a private key is used for data privacy.

51 RSA private
The key format must be 1. Encryption with a private key and decryption with a public key is used for data authentication (e.g. signing).

Reserved
Must be null (binary 0s).

Error Messages

Message ID	Error Message Text
CPF24B4 E	Severe error while addressing parameter list.
CPF3C1E E	Required parameter &1 omitted.
CPF3CF1 E	Error code parameter not valid.
CPF3CF2 E	Error(s) occurred during running of &1 API.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.
CPF9DC2 E	Key-encrypting algorithm context not compatible with key-encrypting key context.
CPF9DC6 E	Algorithm not valid for encrypting or decrypting a key.
CPF9DC8 E	The input data parameter specifies a NULL pointer.
CPF9DC9 E	The total length of data in the input data array is not valid.
CPF9DCE E	A data length is not valid.
CPF9DCF E	A data pointer is not valid.
CPF9DD0 E	Clear data format name not valid.

Message ID	Error Message Text
CPF9DD2 E	Algorithm description format name not valid.
CPF9DD3 E	Key description format name not valid.
CPF9DD4 E	Length of clear data not valid.
CPF9DD6 E	Length of area provided for output data is too small.
CPF9DD7 E	The key-encrypting key context for the specified key is not valid or was previously destroyed.
CPF9DD8 E	The key-encrypting algorithm context for the specified key is not valid or was previously destroyed.
CPF9DD9 E	Effective key size not valid.
CPF9DDA E	Unexpected return code &1.
CPF9DDB E	The key string or Diffie-Hellman parameter string is not valid.
CPF9DDD E	The key string length is not valid.
CPF9DDE E	Cipher algorithm not valid.
CPF9DDF E	Block length not valid.
CPF9DE0 E	Hash algorithm not valid.
CPF9DE1 E	Initialization vector not valid.
CPF9DE2 E	MAC (message authentication code) length not valid.
CPF9DE3 E	Mode not valid.
CPF9DE4 E	Pad option not valid.
CPF9DE5 E	PKA (public key algorithm) block format not valid.
CPF9DE6 E	Public key algorithm not valid.
CPF9DE7 E	Key type not valid.
CPF9DE9 E	Key format not valid.
CPF9DEC E	Cryptographic service provider not valid.
CPF9DED E	Final operation flag not valid.
CPF9DEE E	Reserved field not null.
CPF9DF0 E	Operation, algorithm, or mode not available on the requested CSP (cryptographic service provider).
CPF9DF1 E	The algorithm context token does not reference a valid algorithm context.
CPF9DF2 E	The algorithm context is not found or was previously destroyed.
CPF9DF3 E	Algorithm in algorithm context not valid for requested operation.
CPF9DF4 E	The key context token does not reference a valid key context.
CPF9DF5 E	The key context is not found or was previously destroyed.
CPF9DF7 E	Algorithm context not compatible with key context.
CPF9DF8 E	Cryptographic device name not valid.
CPF9DF9 E	Cryptographic device not found.
CPF9DFA E	Multiple-block encryption not valid with the requested mode.
CPF9DFB E	Cryptographic service provider (CSP) conflicts with the key context CSP.
CPF9DFD E	Not authorized to device.
CPF9DFE E	Cryptographic device not available.
CPF9DFF E	Request not allowed by cryptographic attributes.

API introduced: V5R3

Top | “Cryptographic Services APIs,” on page 1 | APIs by category

Translate Data (QC3TRNDT, Qc3TranslateData)

Required Parameter Group:	
1	Data to translate
Input	Char(*)
2	Length of data to translate
Input	Binary(4)
3	Decrypt algorithm context token
Input	Char(8)
4	Decrypt key context token
Input	Char(8)
5	Encrypt algorithm context token
Input	Char(8)
6	Encrypt key context token
Input	Char(8)
7	Cryptographic service provider
Input	Char(1)
8	Cryptographic device name
Input	Char(10)
9	Translated data
Output	Char(*)
10	Length of area provided for translated data
Input	Binary(4)
11	Length of translated data returned
Output	Binary(4)
12	Error code
I/O	Char(*)
Service Program Name: QC3DTATR	
Default Public Authority: *USE	
Threadsafe: Yes	

The **Translate Data (OPM, QC3TRNDT; ILE, Qc3TranslateData) API** translates data from encryption under one key to encryption under another key.

Authorities and Locks

Required API authority
*USE

Required device description authority
*USE

Required Parameter Group

Data to translate
INPUT; CHAR(*)

The data to be decrypted and encrypted again.

Length of data to translate
INPUT; BINARY(4)

The length of data in the data to translate parameter.
If the decrypt mode of operation is CFB 1-bit, the length must be specified in bits.

Decrypt algorithm context token
INPUT; CHAR(8)

The token for the algorithm context to use for decrypting the data.
The algorithm context is created using the Create Algorithm Context (OPM, QC3CRTAX; ILE, Qc3CreateAlgorithmContext) API.
On a translate operation, the system always performs final processing (e.g. padding) and resets the algorithm context to its initial state. The algorithm context can then be used to begin a new cryptographic operation (encrypt, decrypt, etc.).

Decrypt key context token
INPUT; CHAR(8)

The token for the key context to use for decrypting the data.
The key context is created using the Create Key Context (OPM, QC3CRTKX; ILE, Qc3CreateKeyContext) API.

Encrypt algorithm context token
INPUT; CHAR(8)

The token for the algorithm context to use for encrypting the data.
The algorithm context is created using the Create Algorithm Context (OPM, QC3CRTAX; ILE, Qc3CreateAlgorithmContext) API.

Encrypt key context token
INPUT; CHAR(8)

The token for the key context to use for encrypting the data.
The key context is created using the Create Key Context (OPM, QC3CRTKX; ILE, Qc3CreateKeyContext) API.

Cryptographic service provider
INPUT; CHAR(1)

The cryptographic service provider (CSP) that will perform the translate operation.

0 Any CSP.
The system will choose an appropriate CSP to perform the translate operation.

1 Software CSP.
The system will perform the translate operation using software. If the requested algorithms are not available in software, an error is returned.

2 Hardware CSP.
The system will perform the translate operation using cryptographic hardware. If the requested algorithms are not available in hardware, an error is returned. A specific cryptographic device can be specified using the cryptographic device name parameter. If the cryptographic device is not specified, the system will choose an appropriate one.

Cryptographic device name
INPUT; CHAR(10)

The name of a cryptographic device description.
This parameter is valid when the cryptographic service provider parameter specifies 2 (hardware CSP). Otherwise, this parameter must be blanks or the pointer to this paramter set to NULL.

Translated data
OUTPUT; CHAR(*)

The area to store the translated data.

Length of area provided for translated data
INPUT; BINARY(4)

The length of the translated data parameter.
To ensure sufficient space, specify an area at least as large as the length of data to translate. Be sure to add any space necessary for padding.
If the encrypt mode of operation is CFB 1-bit, this length must be specified in bits.

Length of translated data returned
OUTPUT; BINARY(4)

The length of the translated data returned in the translated data parameter.
If the length of area provided for the translated data is too small, an error will be generated and no data will be returned in the translated data parameter.
If the encrypt mode of operation is CFB 1-bit, the length will be returned in bits.

Error code
I/O; CHAR(*)

The structure in which to return error information.
For the format of the structure, see Error Code Parameter.

Error Messages

Message ID	Error Message Text
CPF24B4 E	Severe error while addressing parameter list.
CPF3C1E E	Required parameter &1 omitted.
CPF3CF1 E	Error code parameter not valid.
CPF3CF2 E	Error(s) occurred during running of &1 API.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.
CPF9DC2 E	Key-encrypting algorithm context not compatible with key-encrypting key context.
CPF9DC3 E	Encrypted data contains invalid padding.
CPF9DC6 E	Algorithm not valid for encrypting or decrypting a key.
CPF9DD4 E	Length of clear data not valid.
CPF9DD5 E	Length of input data not valid.
CPF9DD6 E	Length of area provided for output data is too small.
CPF9DD7 E	The key-encrypting key context for the specified key is not valid or was previously destroyed.
CPF9DD8 E	The key-encrypting algorithm context for the specified key is not valid or was previously destroyed.
CPF9DDA E	Unexpected return code &1.
CPF9DDB E	The key string or Diffie-Hellman parameter string is not valid.
CPF9DDD E	The key string length is not valid.
CPF9DE3 E	Mode not valid.
CPF9DEC E	Cryptographic service provider not valid.
CPF9DF0 E	Operation, algorithm, or mode not available on the requested CSP (cryptographic service provider).
CPF9DF1 E	The algorithm context token does not reference a valid algorithm context.
CPF9DF2 E	The algorithm context is not found or was previously destroyed.
CPF9DF3 E	Algorithm in algorithm context not valid for requested operation.
CPF9DF4 E	The key context token does not reference a valid key context.
CPF9DF5 E	The key context is not found or was previously destroyed.
CPF9DF7 E	Algorithm context not compatible with key context.
CPF9DF8 E	Cryptographic device name not valid.
CPF9DF9 E	Cryptographic device not found.

Message ID	Error Message Text
CPF9DFB E	Cryptographic service provider (CSP) conflicts with the key context CSP.
CPF9DFD E	Not authorized to device.
CPF9DFE E	Cryptographic device not available.
CPF9DFF E	Request not allowed by cryptographic attributes.

API introduced: V5R3

Top | "Cryptographic Services APIs," on page 1 | APIs by category

Authentication APIs

The Authentication APIs allow you to ensure that the following are true:

- Data has not been altered
- Data is not from an impostor

The Authentication APIs include:

- "Calculate Hash (QC3CALHA, Qc3CalculateHash)" on page 23 (QC3CALHA, Qc3CalculateHash) uses a one-way hash function to produce a fixed-length output string from a variable-length input string.
- "Calculate HMAC (QC3CALHM, Qc3CalculateHMAC)" on page 28 (QC3CALHM, Qc3CalculateHMAC) uses a one-way hash function and a secret shared key to produce an authentication value.
- "Calculate MAC (QC3CALMA, Qc3CalculateMAC)" on page 34 (QC3CALMA, Qc3CalculateMAC) produces a message authentication code.
- "Calculate Signature (QC3CALSG, Qc3CalculateSignature)" on page 41 (QC3CALSG, Qc3CalculateSignature) produces a digital signature by hashing the input data and encrypting the hash value using a public key algorithm (PKA).
- "Verify Signature (QC3VFYSG, Qc3VerifySignature)" on page 47 (QC3VFYSG, Qc3VerifySignature) verifies that a digital signature is correctly related to the input data.

Top | Cryptographic Services APIs | APIs by category

Calculate Hash (QC3CALHA, Qc3CalculateHash)

Required Parameter Group:	
1	Input data
Input	Char(*)
2	Length of input data
Input	Binary(4)
3	Input data format name
Input	Char(8)
4	Algorithm description
Input	Char(*)
5	Algorithm description format name
Input	Char(8)
6	Cryptographic service provider
Input	Char(1)
7	Cryptographic device name
Input	Char(10)
8	Hash
Output	Char(*)
9	Error code
I/O	Char(*)
Service Program Name: QC3HASH	
Default Public Authority: *USE	
Threadsafe: Yes	

The **Calculate Hash (OPM, QC3CALHA; ILE, Qc3CalculateHash) API** uses a one-way hash function to produce a fixed-length output string from a variable-length input string. For all practical purposes, one-way hashes are irreversible. This property makes them useful for authentication purposes.

Information on cryptographic standards can be found in the "Create Algorithm Context (QC3CRTAX, Qc3CreateAlgorithmContext)" on page 72 documentation.

Authorities and Locks

Required API authority
*USE

Required device description authority
*USE

Required Parameter Group

Input data
INPUT; CHAR(*)

The data to hash.
The format of the input data is specified in the input data format name parameter

Length of input data
INPUT; BINARY(4)

For input data format DATA0100, this is the length of the data to hash.
For input data format DATA0200, this is the number of entries in the array.

Input data format name
INPUT; CHAR(8)

The format of the input data parameter.
The possible format names follow.

DATA0100
The input data parameter contains the data to hash.

"DATA0200 format" on page 25
The input data parameter contains an array of pointers and lengths to the data to hash. See "Input Data Formats" on page 25 for a description of this format.

Algorithm description
INPUT; CHAR(*)

The algorithm and associated parameters for hashing the data.
The format of the algorithm description is specified in the algorithm description format name parameter.

Algorithm description format name
INPUT; CHAR(8)

The format of the algorithm description.
The possible format names follow.

"ALGD0100 format" on page 25
The token for an algorithm context. This format must be used when performing the hash operation over multiple calls. After the last call (when the final operation flag is on), the context will reset to its initial state and can be used in another API.

"ALGD0500 format" on page 26
Parameters for a hash algorithm (MD5, SHA-1, SHA-256, SHA-384, SHA-512).

See "Algorithm Description Formats" on page 25 for a description of these formats.

Cryptographic service provider
INPUT; CHAR(1)

The cryptographic service provider (CSP) that will perform the hash operation.

0 Any CSP.
The system will choose an appropriate CSP to perform the hash operation.

1 Software CSP.
The system will perform the hash operation using software. If the requested algorithm is not available in software, an error is returned.

2 Hardware CSP.
The system will perform the hash operation using cryptographic hardware. If the requested algorithm is not available in hardware, an error is returned. A specific cryptographic device can be specified using the cryptographic device name parameter. If the cryptographic device is not specified, the system will choose an appropriate one.

Cryptographic device name
INPUT; CHAR(10)

The name of a cryptographic device description.
This parameter is valid when the cryptographic service provider parameter specifies 2 (hardware CSP). Otherwise, this parameter must be blanks or the pointer to this parameter set to NULL.

Hash OUTPUT; CHAR(*)

The area to store the hash. The length of hash is defined by the hash algorithm.

MD5	16 bytes
SHA-1	20 bytes
SHA-256	32 bytes
SHA-384	48 bytes
SHA-512	64 bytes

Error code
I/O; CHAR(*)

The structure in which to return error information.
For the format of the structure, see Error Code Parameter.

Input Data Formats

For detailed descriptions of the table fields, see "**Input Data Formats Field Descriptions**."

DATA0200 format

Offset		Type	Field
Dec	**Hex**		
These fields repeat.		PTR(SPP)	Input data pointer
		BINARY(4)	Input data length
		CHAR(12)	Reserved

Input Data Formats Field Descriptions

Input data length
The length of data to hash.

Input data pointer
A space pointer to the data to hash.

Reserved
Must be null (binary 0s).

Algorithm Description Formats

For detailed descriptions of the table fields, see "**Algorithm Description Formats Field Descriptions**" on page 26.

ALGD0100 format

Offset		Type	Field
Dec	**Hex**		
0	0	CHAR(8)	Algorithm context token
8	8	CHAR(1)	Final operation flag

ALGD0500 format

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Hash algorithm

Algorithm Description Formats Field Descriptions

Algorithm context token

A token for an algorithm context. The algorithm context is created using the "Create Algorithm Context (QC3CRTAX, Qc3CreateAlgorithmContext)" on page 72.

Hash algorithm

The hash algorithm. Following are the valid hash algorithms.

1 MD5
Documented in RFC 1321.

2 SHA-1
Documented in FIPS 180-2.

3 SHA-256
Documented in FIPS 180-2.

4 SHA-384
Documented in FIPS 180-2.

5 SHA-512
Documented in FIPS 180-2.

Final operation flag

The final processing indicator.

0 Continue.
The system will not perform final processing and the algorithm context will maintain the state of the operation. The algorithm context can be used on future calls to this API to continue the hash operation. The pointer to the hash parameter may be set to NULL because the hash value will not be returned until the final operation flag is set on.

1 Final.
The system will perform final processing. The hash value will be returned and the algorithm context will reset to its initial state. The algorithm context can then be used to begin a new cryptographic operation (hash, HMAC, etc.). When performing a final operation, the pointer to the input data parameter may be set to NULL.

Error Messages

Message ID	Error Message Text
CPF24B4 E	Severe error while addressing parameter list.
CPF3C1E E	Required parameter &1 omitted.
CPF3CF1 E	Error code parameter not valid.
CPF3CF2 E	Error(s) occurred during running of &1 API.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.
CPF9DC7 E	The output data parameter specifies a NULL pointer.
CPF9DC8 E	The input data parameter specifies a NULL pointer.
CPF9DC9 E	The total length of data in the input data array is not valid.
CPF9DCE E	A data length is not valid.
CPF9DCF E	A data pointer is not valid.
CPF9DD1 E	Input data format name not valid.
CPF9DD2 E	Algorithm description format name not valid.
CPF9DD5 E	Length of input data not valid.
CPF9DDA E	Unexpected return code &1.
CPF9DE0 E	Hash algorithm not valid.

Message ID	Error Message Text
CPF9DEC E	Cryptographic service provider not valid.
CPF9DED E	Final operation flag not valid.
CPF9DEE E	Reserved field not null.
CPF9DF0 E	Operation, algorithm, or mode not available on the requested CSP (cryptographic service provider).
CPF9DF1 E	The algorithm context token does not reference a valid algorithm context.
CPF9DF2 E	The algorithm context is not found or was previously destroyed.
CPF9DF3 E	Algorithm in algorithm context not valid for requested operation.
CPF9DF8 E	Cryptographic device name not valid.
CPF9DF9 E	Cryptographic device not found.
CPF9DFD E	Not authorized to device.
CPF9DFE E	Cryptographic device not available.

API introduced: V5R3

Top | "Cryptographic Services APIs," on page 1 | APIs by category

Calculate HMAC (QC3CALHM, Qc3CalculateHMAC)

Required Parameter Group:	
1	Input data
Input	Char(*)
2	Length of input data
Input	Binary(4)
3	Input data format name
Input	Char(8)
4	Algorithm description
Input	Char(*)
5	Algorithm description format name
Input	Char(8)
6	Key description
Input	Char(*)
7	Key description format name
Input	Char(8)
8	Cryptographic service provider
Input	Char(1)
9	Cryptographic device name
Input	Char(10)
10	HMAC
Output	Char(*)
11	Error code
I/O	Char(*)
Service Program Name: QC3HMAC	
Default Public Authority: *USE	
Threadsafe: Yes	

The **Calculate HMAC (OPM, QC3CALHM; ILE Qc3CalculateHMAC) API** uses a one-way hash function and a secret shared key to produce an authentication value. The HMAC function is documented in RFC 2104.

Information on cryptographic standards can be found in the “Create Algorithm Context (QC3CRTAX, Qc3CreateAlgorithmContext)” on page 72 documentation.

Authorities and Locks

Required API authority
*USE

Required device description authority
*USE

Required Parameter Group

Input data
INPUT; CHAR(*)

The data to hash.
The format of the input data is specified in the input data format name parameter

Length of input data
INPUT; BINARY(4)

For input data format DATA0100, this is the length of the data to hash.
For input data format DATA0200, this is the number of entries in the array.

Input data format name
INPUT; CHAR(8)

The format of the input data parameter.
The possible format names follow.

DATA0100
The input data parameter contains the data to hash.

"DATA0200 format" on page 30
The input data parameter contains an array of pointers and lengths to the data to hash. See "Input Data Formats" on page 30 for a description of this format.

Algorithm description
INPUT; CHAR(*)

The algorithm and associated parameters for hashing the data.
The format of the algorithm description is specified in the algorithm description format name parameter.

Algorithm description format name
INPUT; CHAR(8)

The format of the algorithm description.
The possible format names follow.

"ALGD0100 format" on page 31
The token for an algorithm context. This format must be used when performing the HMAC operation over multiple calls. After the last call (when the final operation flag is on), the context will reset to its initial state and can be used in another API.

"ALGD0500 format" on page 31
Parameters for a hash algorithm (MD5 or SHA-1).

See "Algorithm Description Formats" on page 31 for a description of these formats.

Key description
INPUT; CHAR(*)

The key and associated parameters for the HMAC operation.
The format of the key description is specified in the key description format name parameter.
If the HMAC operation extends over multiple calls (see ALGD0100 description above), only the key description from the first call will be used. Therefore, on subsequent calls, you may set the pointer to this parameter to NULL.

Key description format name
INPUT; CHAR(8)

The format of the key description.
If the pointer to the key description parameter is NULL, this parameter will be ignored.
The possible format names follow.

"KEYD0100 format" on page 32
The token for a key context. This format must be used if the key is encrypted.

"KEYD0200 format" on page 32
Key parameters.

See "Key Description Formats" on page 32 for a description of these formats.

Cryptographic service provider
INPUT; CHAR(1)

The cryptographic service provider (CSP) that will perform the decryption operation.

0 Any CSP.
The system will choose an appropriate CSP to perform the HMAC operation.

1 Software CSP.
The system will perform the HMAC operation using software. If the requested algorithm is not available in software, an error is returned.

2 Hardware CSP.
The system will perform the HMAC operation using cryptographic hardware. If the requested algorithm is not available in hardware, an error is returned. A specific cryptographic device can be specified using the cryptographic device name parameter. If the cryptographic device is not specified, the system will choose an appropriate one.

Cryptographic device name
INPUT; CHAR(10)

The name of a cryptographic device description.
This parameter is valid when the cryptographic service provider parameter specifies 2 (hardware CSP). Otherwise, this parameter must be blanks or the pointer to this parameter set to NULL.

HMAC
OUTPUT; CHAR(*)

The area to store the HMAC. The length of HMAC is defined by the hash algorithm.

MD5 16 bytes
SHA-1 20 bytes

Error code
I/O; CHAR(*)

The structure in which to return error information.
For the format of the structure, see Error Code Parameter.

Input Data Formats

For detailed descriptions of the table fields, see "**Input Data Formats Field Descriptions**" on page 31.

DATA0200 format

Offset		Type	Field
Dec	Hex		
These fields repeat.		PTR(SPP)	Input data pointer
		BINARY(4)	Input data length
		CHAR(12)	Reserved

Input Data Formats Field Descriptions

Input data length
The length of data to hash.

Input data pointer
A space pointer to the data to hash.

Reserved
Must be null (binary 0s).

Algorithm Description Formats

For detailed descriptions of the table fields, see "**Algorithm Description Formats Field Descriptions**."

ALGD0100 format

Offset		Type	Field
Dec	Hex		
0	0	CHAR(8)	Algorithm context token
8	8	CHAR(1)	Final operation flag

ALGD0500 format

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Hash algorithm

Algorithm Description Formats Field Descriptions

Algorithm context token
A token for an algorithm context. The algorithm context is created using the "Create Algorithm Context (QC3CRTAX, Qc3CreateAlgorithmContext)" on page 72.

Final operation flag
The final processing indicator.

0 Continue.
The system will not perform final processing and the algorithm context will maintain the state of the operation. The algorithm context can be used on future calls to this API to continue the HMAC operation. The pointer to the HMAC parameter may be set to NULL because the HMAC value will not be returned until the final operation flag is set on.

1 Final.
The system will perform final processing. The HMAC value will be returned and the algorithm context will reset to its initial state. The algorithm context can then be used to begin a new cryptographic operation (hash, HMAC etc.). When performing a final operation, the pointer to the input data parameter may be set to NULL and the length of input data parameter set to 0.

Hash algorithm
The hash algorithm. Following are the valid hash algorithms.

1 MD5
Documented in RFC 1321.

2 SHA-1
Documented in FIPS 180-2.

Key Description Formats

For detailed descriptions of the table fields, see "**Key Description Formats Field Descriptions**."

KEYD0100 format

Offset		Type	Field
Dec	**Hex**		
0	0	CHAR(8)	Key context token

KEYD0200 format

Offset		Type	Field
Dec	**Hex**		
0	0	BINARY(4)	Key type
4	4	BINARY(4)	Key string length
8	8	CHAR(1)	Key format
9	9	CHAR(3)	Reserved
12	C	CHAR(*)	Key string

Key Description Formats Field Descriptions

Key context token

A token for a key context. The key context is created using the "Create Key Context (QC3CRTKX, Qc3CreateKeyContext)" on page 78.

Key format

The format of the key string field. Following are the valid values.

0 Binary string.
The key is specified as a binary value. To obtain a good random key value, use the "Generate Pseudorandom Numbers (QC3GENRN,Qc3GenPRNs) API" on page 70.

Key string

The key to use in the HMAC operation.

Key string length

Length of the key string specified in the key string field. Refer to the key type field for more information.

Key type

The type of key. Following are the valid values.

1 MD5
The minimum length for an MD5 HMAC key is 16 bytes.

2 SHA-1
The minimum length for an SHA-1 HMAC key is 20 bytes.

A key longer than the minimum length does not significantly increase the function strength unless the randomness of the key is considered weak. A key longer than 64 bytes will be hashed before it is used.

Reserved

Must be null (binary 0s).

Error Messages

Message ID	Error Message Text
CPF24B4 E	Severe error while addressing parameter list.
CPF3C1E E	Required parameter &1 omitted.
CPF3CF1 E	Error code parameter not valid.
CPF3CF2 E	Error(s) occurred during running of &1 API.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.
CPF9DC2 E	Key-encrypting algorithm context not compatible with key-encrypting key context.
CPF9DC6 E	Algorithm not valid for encrypting or decrypting a key.
CPF9DC9 E	The total length of data in the input data array is not valid.
CPF9DCE E	A data length is not valid.
CPF9DCF E	A data pointer is not valid.
CPF9DD1 E	Input data format name not valid.
CPF9DD2 E	Algorithm description format name not valid.
CPF9DD3 E	Key description format name not valid.
CPF9DD5 E	Length of input data not valid.
CPF9DD7 E	The key-encrypting key context for the specified key is not valid or was previously destroyed.
CPF9DD8 E	The key-encrypting algorithm context for the specified key is not valid or was previously destroyed.
CPF9DDA E	Unexpected return code &1.
CPF9DDB E	The key string or Diffie-Hellman parameter string is not valid.
CPF9DDD E	The key string length is not valid.
CPF9DE0 E	Hash algorithm not valid.
CPF9DE7 E	Key type not valid.
CPF9DE9 E	Key format not valid.
CPF9DEC E	Cryptographic service provider not valid.
CPF9DED E	Final operation flag not valid.
CPF9DEE E	Reserved field not null.
CPF9DF0 E	Operation, algorithm, or mode not available on the requested CSP (cryptographic service provider).
CPF9DF1 E	The algorithm context token does not reference a valid algorithm context.
CPF9DF2 E	The algorithm context is not found or was previously destroyed.
CPF9DF3 E	Algorithm in algorithm context not valid for requested operation.
CPF9DF4 E	The key context token does not reference a valid key context.
CPF9DF5 E	The key context is not found or was previously destroyed.
CPF9DF7 E	Algorithm context not compatible with key context.
CPF9DF8 E	Cryptographic device name not valid.
CPF9DF9 E	Cryptographic device not found.
CPF9DFB E	Cryptographic service provider (CSP) conflicts with the key context CSP.
CPF9DFD E	Not authorized to device.
CPF9DFE E	Cryptographic device not available.

API introduced: V5R3

Top | "Cryptographic Services APIs," on page 1 | APIs by category

Calculate MAC (QC3CALMA, Qc3CalculateMAC)

Required Parameter Group:	
1	Input data
Input	Char(*)
2	Length of input data
Input	Binary(4)
3	Input data format name
Input	Char(8)
4	Algorithm description
Input	Char(*)
5	Algorithm description format name
Input	Char(8)
6	Key description
Input	Char(*)
7	Key description format name
Input	Char(8)
8	Cryptographic service provider
Input	Char(1)
9	Cryptographic device name
Input	Char(10)
10	MAC
Output	Char(*)
11	Error code
I/O	Char(*)
Service Program Name: QC3MAC	
Default Public Authority: *USE	
Threadsafe: Yes	

The **Calculate MAC (OPM, QC3CALMA; ILE, Qc3CalculateMAC) API** produces a message authentication code. Normally, a MAC is appended to the end of a message and later used to check the message's integrity. To produce a MAC, the input data is encrypted using CBC (cipher block chaining) mode. Some or all of the bytes from the last encrypted data block are returned as the MAC value.

Information on cryptographic standards can be found in the "Create Algorithm Context (QC3CRTAX, Qc3CreateAlgorithmContext)" on page 72 documentation.

Authorities and Locks

Required API authority
*USE

Required device description authority
*USE

Required Parameter Group

Input data
INPUT; CHAR(*)

The data to encrypt.
The format of the input data is specified in the input data format name parameter

Length of input data
INPUT; BINARY(4)

For input data format DATA0100, this is the length of the data to encrypt. If it is not a multiple of the block length, the data will be padded with hex 00s.

For input data format DATA0200, this is the number of entries in the array.

Input data format name
INPUT; CHAR(8)

The format of the input data parameter.
The possible format names follow.

DATA0100
The input data parameter contains the data to encrypt.

"DATA0200 format" on page 36
The input data parameter contains an array of pointers and lengths to the data to encrypt.
See "Input Data Formats" on page 36 for a description of this format.

Algorithm description
INPUT; CHAR(*)

The algorithm and associated parameters for encrypting the data.
The format of the algorithm description is specified in the algorithm description format name parameter.

Algorithm description format name
INPUT; CHAR(8)

The format of the algorithm description.
The possible format names follow.

"ALGD0100 format" on page 37
The token for an algorithm context. This format must be used when performing the MAC operation over multiple calls. After the last call (when the final operation flag is on), the context will reset to its initial state and can be used in another API.

"ALGD0200 format" on page 37
Parameters for a block cipher algorithm (DES, Triple DES, and AES).

See "Algorithm Description Formats" on page 37 for a description of these formats.

Key description
INPUT; CHAR(*)

The key and associated parameters for encrypting the data.
The format of the key description is specified in the key description format name parameter.
If the MAC operation extends over multiple calls (see ALGD0100 description above), only the key description from the first call will be used. Therefore, on subsequent calls, you may set the pointer to this parameter to NULL.

Key description format name
INPUT; CHAR(8)

The format of the key description.
If the pointer to the key description parameter is NULL, this parameter will be ignored.
The possible format names follow.

"KEYD0100 format" on page 38
The token for a key context. This format must be used if the key is encrypted.

"KEYD0200 format" on page 39
Key parameters.

See "Key Description Formats" on page 38 for a description of these formats.

Cryptographic service provider
INPUT; CHAR(1)

The cryptographic service provider (CSP) that will perform the decryption operation.

0 Any CSP.
The system will choose an appropriate CSP to perform the MAC operation.

1 Software CSP.
The system will perform the MAC operation using software. If the requested algorithm is not available in software, an error is returned.

2 Hardware CSP.
The system will perform the MAC operation using cryptographic hardware. If the requested algorithm is not available in hardware, an error is returned. A specific cryptographic device can be specified using the cryptographic device name parameter. If the cryptographic device is not specified, the system will choose an appropriate one.

Cryptographic device name
INPUT; CHAR(10)

The name of a cryptographic device description.
This parameter is valid when the cryptographic service provider parameter specifies 2 (hardware CSP). Otherwise, this parameter must be blanks or the pointer to this parameter set to NULL.

MAC OUTPUT; CHAR(*)

The area to store the MAC. The length of MAC is specified in the MAC length field in the algorithm description.

Error code
I/O; CHAR(*)

The structure in which to return error information.
For the format of the structure, see Error Code Parameter.

Input Data Formats

For detailed descriptions of the table fields, see "**Input Data Formats Field Descriptions**" on page 37.

DATA0200 format

Offset		Type	Field
Dec	**Hex**		
These fields repeat.		PTR(SPP)	Input data pointer
		BINARY(4)	Input data length
		CHAR(12)	Reserved

Input Data Formats Field Descriptions

Input data length
The length of data to encrypt. When final processing is performed and the total of all the input data lengths is not a multiple of the block length, the data will be padded with hex 00s.

Input data pointer
A space pointer to the data to encrypt.

Reserved
Must be null (binary 0s).

Algorithm Description Formats

For detailed descriptions of the table fields, see "**Algorithm Description Formats Field Descriptions**."

ALGD0100 format

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	CHAR(8)	Algorithm context token
8	8	CHAR(1)	Final operation flag

ALGD0200 format

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	BINARY(4)	Block cipher algorithm
4	4	BINARY(4)	Block length
8	8	CHAR(1)	Mode
9	9	CHAR(1)	Pad option
10	A	CHAR(1)	Pad character
11	B	CHAR(1)	Reserved
12	C	BINARY(4)	MAC length
16	10	BINARY(4)	Effective key size
20	14	CHAR(32)	Initialization vector

Algorithm Description Formats Field Descriptions

Algorithm context token
A token for an algorithm context. The algorithm context is created using the "Create Algorithm Context (QC3CRTAX, Qc3CreateAlgorithmContext)" on page 72.

Block cipher algorithm
The encryption algorithm. Following are the valid block cipher algorithms.

20 DES
21 Triple DES
22 AES

Block length
The algorithm block length. For DES and Triple DES this field must specify 8. The valid block length values for AES are 16, 24, and 32. Note: 16 is the only block length specified in the AES standard, FIPS-197.

Effective key size
Effective key size is not used on a MAC operation and must be set to null (binary 0's).

Final operation flag
The final processing indicator.

0 Continue.
The system will not perform final processing and the algorithm context will maintain the state of the operation. The algorithm context can be used on future calls to this API to continue the MAC operation. The pointer to the MAC parameter may be set to NULL because the MAC value will not be returned until the final operation flag is set on.

1 Final.
The system will perform final processing (e.g. padding). The MAC value will be returned and the algorithm context will reset to its initial state. The algorithm context can then be used to begin a new cryptographic operation (encrypt, decrypt, etc.). When performing a final operation, the pointer to the input data parameter may be set to NULL and the length of the input data parameter set to 0.

Initialization vector
The initialization vector (IV). For an explanation of its use, refer to the mode standards for CBC in FIPS PUB 81 and ANSI X9.52. For DES and Triple DES, the first 8 bytes are used as the IV. For AES, the length of IV used is that specified by block length. The IV need not be secret, but it should be unique for each message. If not unique, it may compromise security. The IV can be any value. To obtain a good random IV value, use the "Generate Pseudorandom Numbers (QC3GENRN,Qc3GenPRNs) API" on page 70.

MAC length
The message authentication code length. It can not exceed the block length value. The leftmost MAC length bytes from the last block of encrypted data are returned as the MAC.

Mode The mode of operation. Information on modes can be found in FIPS PUB 81 and ANSI X9.52. Following are the valid modes for a MAC operation.

1 CBC

Pad character
This field is not used on a MAC operation and must be set to null (binary 0s).

Pad option
Following are the valid pad options for a MAC operation.

0 If the length of input data is not a multiple of 8, the input data will be padded with null (binary 0s).

Reserved
Must be null (binary 0s).

Key Description Formats

For detailed descriptions of the table fields, see "**Key Description Formats Field Descriptions**" on page 39.

KEYD0100 format

Offset		Type	Field
Dec	Hex		
0	0	CHAR(8)	Key context token

KEYD0200 format

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Key type
4	4	BINARY(4)	Key string length
8	8	CHAR(1)	Key format
9	9	CHAR(3)	Reserved
12	C	CHAR(*)	Key string

Key Description Formats Field Descriptions

Key context token

A token for a key context. The key context is created using the "Create Key Context (QC3CRTKX, Qc3CreateKeyContext)" on page 78.

Key format

The format of the key string field. Following are the valid values.

0 Binary string.
The key is specified as a binary value. To obtain a good random key value, use the "Generate Symmetric Key (QC3GENSK, Qc3GenSymmetricKey)" on page 65, or "Generate Pseudorandom Numbers (QC3GENRN,Qc3GenPRNs) API" on page 70 API.

Key string

The key to use in the MAC operation.

Key string length

Length of the key string specified in the key string field.

Key type

The type of key. Following are the valid values.

20 DES
The key format must be 0. The key string must be 8 bytes in length. Only 7 bits of each byte are used as the actual key. The rightmost bit of each byte is used to set parity. Some cryptographic service providers require that a DES key have odd parity in every byte. Others ignore parity.

21 Triple DES
The key format must be 0. The key string can be 8, 16, or 24 bytes in length. When 24 bytes are specified, the first 8 bytes are used for key 1, the second 8 bytes for key 2, and the third 8 bytes for key 3. When 16 bytes are specified the first 8 bytes are used for keys 1 and 3, and the second 8 bytes for key 2. When just 8 bytes are specified, the first 8 bytes are used for all 3 keys. A MAC operation using Triple DES encrypts the entire input data (plus any padding) using DES and key 1. The last block is then decrypted using key 2 and encrypted again with key 3. Only 7 bits of each byte are used as the actual key. The rightmost bit of each byte is used to set parity. Some cryptographic service providers require that a Triple DES key have odd parity in every byte. Others ignore parity.

22 AES
The key format must be 0. The key string can be 16, 24, or 32 bytes in length.

Reserved

Must be null (binary 0s).

Error Messages

Message ID	Error Message Text
CPF24B4 E	Severe error while addressing parameter list.
CPF3C1E E	Required parameter &1 omitted.
CPF3CF1 E	Error code parameter not valid.
CPF3CF2 E	Error(s) occurred during running of &1 API.

Message ID	Error Message Text
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.
CPF9DC2 E	Key-encrypting algorithm context not compatible with key-encrypting key context.
CPF9DC6 E	Algorithm not valid for encrypting or decrypting a key.
CPF9DC9 E	The total length of data in the input data array is not valid.
CPF9DCD E	Pad character not valid.
CPF9DCE E	A data length is not valid.
CPF9DCF E	A data pointer is not valid.
CPF9DD0 E	Clear data format name not valid.
CPF9DD2 E	Algorithm description format name not valid.
CPF9DD3 E	Key description format name not valid.
CPF9DD5 E	Length of input data not valid.
CPF9DD6 E	Length of area provided for output data is too small.
CPF9DD7 E	The key-encrypting key context for the specified key is not valid or was previously destroyed.
CPF9DD8 E	The key-encrypting algorithm context for the specified key is not valid or was previously destroyed.
CPF9DD9 E	Effective key size not valid.
CPF9DDA E	Unexpected return code &1.
CPF9DDB E	The key string or Diffie-Hellman parameter string is not valid.
CPF9DDD E	The key string length is not valid.
CPF9DDE E	Cipher algorithm not valid.
CPF9DDF E	Block length not valid.
CPF9DE2 E	MAC (message authentication code) length not valid.
CPF9DE3 E	Mode not valid.
CPF9DE4 E	Pad option not valid.
CPF9DE7 E	Key type not valid.
CPF9DE9 E	Key format not valid.
CPF9DEC E	Cryptographic service provider not valid.
CPF9DED E	Final operation flag not valid.
CPF9DEE E	Reserved field not null.
CPF9DF0 E	Operation, algorithm, or mode not available on the requested CSP (cryptographic service provider).
CPF9DF1 E	The algorithm context token does not reference a valid algorithm context.
CPF9DF2 E	The algorithm context is not found or was previously destroyed.
CPF9DF3 E	Algorithm in algorithm context not valid for requested operation.
CPF9DF4 E	The key context token does not reference a valid key context.
CPF9DF5 E	The key context is not found or was previously destroyed.
CPF9DF7 E	Algorithm context not compatible with key context.
CPF9DF8 E	Cryptographic device name not valid.
CPF9DF9 E	Cryptographic device not found.
CPF9DFB E	Cryptographic service provider (CSP) conflicts with the key context CSP.
CPF9DFD E	Not authorized to device.
CPF9DFE E	Cryptographic device not available.

API introduced: V5R3

Top | "Cryptographic Services APIs," on page 1 | APIs by category

Calculate Signature (QC3CALSG, Qc3CalculateSignature)

Required Parameter Group:

1	Input data
Input	Char(*)
2	Length of input data
Input	Binary(4)
3	Input data format name
Input	Char(8)
4	Algorithm description
Input	Char(*)
5	Algorithm description format name
Input	Char(8)
6	Key description
Input	Char(*)
7	Key description format name
Input	Char(8)
8	Cryptographic service provider
Input	Char(1)
9	Cryptographic device name
Input	Char(10)
10	Signature
Output	Char(*)
11	Length of area provided for signature
Input	Binary(4)
12	Length of signature returned
Output	Binary(4)
13	Error code
I/O	Char(*)

Service Program Name: QC3SIGCL

Default Public Authority: *USE

Threadsafe: Yes

The **Calculate Signature (OPM, QC3CALSG; ILE, Qc3CalculateSignature) API** produces a digital signature by hashing the input data and encrypting the hash value using a public key algorithm (PKA). To verify the signature, use the "Verify Signature (QC3VFYSG, Qc3VerifySignature)" on page 47.

Information on cryptographic standards can be found in the "Create Algorithm Context (QC3CRTAX, Qc3CreateAlgorithmContext)" on page 72 documentation.

Authorities and Locks

Required API authority
*USE

Required device description authority
*USE

Required Parameter Group

Input data
INPUT; CHAR(*)

The data to sign.
The format of the input data is specified in the input data format name parameter

Length of input data
INPUT; BINARY(4)

For input data format DATA0100, this is the length of the data to sign.
For input data format DATA0200, this is the number of entries in the array.

Input data format name
INPUT; CHAR(8)

The format of the input data parameter.
The possible format names follow.

DATA0100
The input data parameter contains the data to sign.

"DATA0200 format" on page 44
The input data parameter contains an array of pointers and lengths to the data to sign.
See "Input Data Formats" on page 44 for a description of this format.

Algorithm description
INPUT; CHAR(*)

The algorithm and associated parameters for signing the data.
The format of the algorithm description is specified in the algorithm description format name parameter.

Algorithm description format name
INPUT; CHAR(8)

The format of the algorithm description.
The possible format names follow.

"ALGD0100 format" on page 44
The token for an algorithm context. This format must be used when performing the sign operation over multiple calls. After the last call (when the final operation flag is on), the context will reset to its initial state and can be used in another API.

"ALGD0400 format" on page 44
Parameters for a sign operation.

See "Algorithm Description Formats" on page 44 for a description of these formats.

Key description
INPUT; CHAR(*)

The key and associated parameters for signing the data.
The format of the key description is specified in the key description format name parameter.
If the sign operation extends over multiple calls (see ALGD0100 description above), only the key description from the first call will be used. Therefore, on subsequent calls, you may set the pointer to this parameter to NULL.

Key description format name
INPUT; CHAR(8)

The format of the key description.
If the pointer to the key description parameter is NULL, this parameter will be ignored.
The possible format names follow.

"KEYD0100 format" on page 45
The token for a key context. This format must be used if the key is encrypted.

"KEYD0200 format" on page 45
Key parameters.

See "Key Description Formats" on page 45 for a description of these formats.

Cryptographic service provider
INPUT; CHAR(1)

The cryptographic service provider (CSP) that will perform the sign operation.

0 Any CSP.
The system will choose an appropriate CSP to perform the sign operation.

1 Software CSP.
The system will perform the sign operation using software. If the requested algorithm is not available in software, an error is returned.

2 Hardware CSP.
The system will perform the sign operation using cryptographic hardware. If the requested algorithm is not available in hardware, an error is returned. A specific cryptographic device can be specified using the cryptographic device name parameter. If the cryptographic device is not specified, the system will choose an appropriate one.

Cryptographic device name
INPUT; CHAR(10)

The name of a cryptographic device description.
This parameter is valid when the cryptographic service provider parameter specifies 2 (hardware CSP). Otherwise, this parameter must be blanks or the pointer to this parameter set to NULL.

Signature
OUTPUT; CHAR(*)

The area to store the signature.

Length of area provided for signature
INPUT; BINARY(4)

The length of the signature parameter in bytes. The length of the signature will equal the key size. (See "Generate PKA Key Pair (QC3GENPK, Qc3GenPKAKeyPair)" on page 61.) Because key size is normally specified in bits, divide that value by 8 and round up to obtain the length of area needed for the signature.

Length of signature returned
OUTPUT; BINARY(4)

The length of the signature returned in the signature parameter.
If the length of area provided for the signature is too small, an error will be generated and no data will be returned in the signature parameter.

Error code
I/O; CHAR(*)

The structure in which to return error information.
For the format of the structure, see Error Code Parameter.

Input Data Formats

For detailed descriptions of the table fields, see "**Input Data Formats Field Descriptions**."

DATA0200 format

Offset			
Dec	**Hex**	**Type**	**Field**
These fields repeat.		PTR(SPP)	Input data pointer
		BINARY(4)	Input data length
		CHAR(12)	Reserved

Input Data Formats Field Descriptions

Input data length
The length of data to sign.

Input data pointer
A space pointer to the data to sign.

Reserved
Must be null (binary 0s).

Algorithm Description Formats

For detailed descriptions of the table fields, see "**Algorithm Description Formats Field Descriptions**."

ALGD0100 format

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	CHAR(8)	Algorithm context token
8	8	CHAR(1)	Final operation flag

ALGD0400 format

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	BINARY(4)	Public key cipher algorithm
4	4	CHAR(1)	PKA block format
5	5	CHAR(3)	Reserved
8	8	BINARY(4)	Signing hash algorithm

Algorithm Description Formats Field Descriptions

Algorithm context token
A token for an algorithm context. The algorithm context is created using the "Create Algorithm Context (QC3CRTAX, Qc3CreateAlgorithmContext)" on page 72.

Final operation flag
The final processing indicator.

0 Continue.
The system will not perform final processing and the algorithm context will maintain the state of the operation. The algorithm context can be used on future calls to this API to continue the sign operation. The signature will not be returned until the final operation flag is set on. The pointer to the signature parameter may be set to NULL because the signature will not be returned until the final operation flag is set on.

1 Final.
The system will perform final processing. The signature will be returned and the algorithm context will reset to its initial state. The algorithm context can then be used to begin a new cryptographic operation. When performing a final operation, the pointer to the input data parameter may be set to NULL.

PKA block format
The public key algorithm block format. Following are the valid values.

0 PKCS #1 block type 00

1 PKCS #1 block type 01

3 ISO 9796-1
Because of security weaknesses, this format should be used for compatibility purposes only.

5 ANSI X9.31
This format is only valid with signing hash algorithm 2 (SHA-1).

Public key cipher algorithm
The encryption algorithm. Following are the valid public key cipher algorithms.

50 RSA

Reserved
Must be null (binary 0s).

Signing hash algorithm
The hash algorithm. Following are the valid values for the signing hash algorithm.

1 MD5

2 SHA-1

Key Description Formats

For detailed descriptions of the table fields, see "**Key Description Formats Field Descriptions**" on page 46.

KEYD0100 format

Offset		Type	Field
Dec	**Hex**		
0	0	CHAR(8)	Key context token

KEYD0200 format

Offset		Type	Field
Dec	**Hex**		
0	0	BINARY(4)	Key type
4	4	BINARY(4)	Key string length
8	8	CHAR(1)	Key format
9	9	CHAR(3)	Reserved
12	C	CHAR(*)	Key string

Key Description Formats Field Descriptions

Key context token

A token for a key context. The key context is created using the "Create Key Context (QC3CRTKX, Qc3CreateKeyContext)" on page 78.

Key format

The format of the key string field. Following are the valid values.

1 BER string
The key is specified in BER encoded PKCS #8 format. For specifications of this format, refer to RSA Security Inc. Public-Key Cryptography Standards. To generate a PKA key pair in this format, use the "Generate PKA Key Pair (QC3GENPK, Qc3GenPKAKeyPair)" on page 61.

Key string

The key to use in the sign operation.

Key string length

Length of the key string specified in the key string field.

Key type

The type of key. Following are the valid values.

51 RSA private

Reserved

Must be null (binary 0s).

Error Messages

Message ID	Error Message Text
CPF24B4 E	Severe error while addressing parameter list.
CPF3C1E E	Required parameter &1 omitted.
CPF3CF1 E	Error code parameter not valid.
CPF3CF2 E	Error(s) occurred during running of &1 API.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.
CPF9DC2 E	Key-encrypting algorithm context not compatible with key-encrypting key context.
CPF9DC6 E	Algorithm not valid for encrypting or decrypting a key.
CPF9DC9 E	The total length of data in the input data array is not valid.
CPF9DCC E	The length of area provided for signature is not valid.
CPF9DCE E	A data length is not valid.
CPF9DCF E	A data pointer is null.
CPF9DD0 E	Clear data format name not valid.
CPF9DD2 E	Algorithm description format name not valid.
CPF9DD3 E	Key description format name not valid.
CPF9DD5 E	Length of input data not valid.
CPF9DD6 E	Length of area provided for output data is too small.
CPF9DD7 E	The key-encrypting key context for the specified key is not valid or was previously destroyed.
CPF9DD8 E	The key-encrypting algorithm context for the specified key is not valid or was previously destroyed.
CPF9DDA E	Unexpected return code &1.
CPF9DDB E	The key string or Diffie-Hellman parameter string is not valid.
CPF9DDD E	The key string length is not valid.
CPF9DE0 E	Hash algorithm not valid.
CPF9DE3 E	Mode not valid.
CPF9DE5 E	PKA (public key algorithm) block format not valid.
CPF9DE6 E	Public key algorithm not valid.
CPF9DE7 E	Key type not valid.
CPF9DE9 E	Key format not valid.
CPF9DEC E	Cryptographic service provider not valid.
CPF9DED E	Final operation flag not valid.

Message ID	Error Message Text
CPF9DEE E	Reserved field not null.
CPF9DF0 E	Operation, algorithm, or mode not available on the requested CSP (cryptographic service provider).
CPF9DF1 E	The algorithm context token does not reference a valid algorithm context.
CPF9DF2 E	The algorithm context is not found or was previously destroyed.
CPF9DF3 E	Algorithm in algorithm context not valid for requested operation.
CPF9DF4 E	The key context token does not reference a valid key context.
CPF9DF5 E	The key context is not found or was previously destroyed.
CPF9DF7 E	Algorithm context not compatible with key context.
CPF9DF8 E	Cryptographic device name not valid.
CPF9DF9 E	Cryptographic device not found.
CPF9DFB E	Cryptographic service provider conflicts with the key context CSP.
CPF9DFD E	Not authorized to device.
CPF9DFE E	Cryptographic device not available.

API introduced: V5R3

Top | "Cryptographic Services APIs," on page 1 | APIs by category

---

Verify Signature (QC3VFYSG, Qc3VerifySignature)

Required Parameter Group:

1	Signature	Input	Char(*)
2	Length of signature	Input	Binary(4)
3	Input data	Input	Char(*)
4	Length of input data	Input	Binary(4)
5	Input data format name	Input	Char(8)
6	Algorithm description	Input	Char(*)
7	Algorithm description format name	Input	Char(8)
8	Key description	Input	Char(*)
9	Key description format name	Input	Char(8)
10	Cryptographic service provider	Input	Char(1)
11	Cryptographic device name	Input	Char(10)
12	Error code	I/O	Char(*)

Service Program Name: QC3SIGVR

Default Public Authority: *USE

Threadsafe: Yes

The **Verify Signature (OPM, QC3VFYSG; ILE, Qc3VerifySignature) API** verifies a digital signature is correctly related to the input data. If the verification fails with a CPF9DEF, the input data has been corrupted. A digital signature is created by hashing data and encrypting the hash value using a public key algorithm (PKA). A digital signature can be created using the Calculate Signature (OPM, QC3CALSG; ILE, Qc3CalculateSignature) API.

Authorities and Locks

Required API authority
*USE

Required device description authority
*USE

Required Parameter Group

Signature
INPUT; CHAR(*)

The digital signature to verify.

Length of signature
INPUT; BINARY(4)

The length of signature should be equal to the key size (size of the modulus), but expressed in bytes.

Input data
INPUT; CHAR(*)

The data to verify.
The format of the input data is specified in the input data format name parameter.

Length of input data
INPUT; BINARY(4)

For input data format DATA0100, this is the length of the data to verify.
For input data format DATA0200, this is the number of entries in the array.

Input data format name
INPUT; CHAR(8)

The format of the input data parameter.
The possible format names follow.

DATA0100
The input data parameter contains the data to verify.

"DATA0200 format" on page 49
The input data parameter contains an array of pointers and lengths to the data to verify. See "Input Data Formats" on page 49 for a description of this format.

Algorithm description
INPUT; CHAR(*)

The algorithm and associated parameters for verifying the data.
The format of the algorithm description is specified in the algorithm description format name parameter.

Algorithm description format name
INPUT; CHAR(8)

The format of the algorithm description.
The possible format names follow.

"ALGD0100 format" on page 50
The token for an algorithm context. This format must be used when performing the verify signature operation over multiple calls. After the last call (when the final operation flag is on), the context will reset to its initial state and can be used in another API.

"ALGD0400 format" on page 50
Parameters for a verify signature operation.

See "Algorithm Description Formats" on page 50 for a description of these formats.

Key description
INPUT; CHAR(*)

The key and associated parameters for verifying the data.
The format of the key description is specified in the key description format name parameter.
If the verify operation extends over multiple calls (see ALGD0100 description above), only the key description from the first call will be used. Therefore, on subsequent calls, you may set the pointer to this parameter to NULL.

Key description format name
INPUT; CHAR(8)

The format of the key description.
If the pointer to the key description parameter is NULL, this parameter will be ignored.
The possible format names follow.

"KEYD0100 format" on page 51
The token for a key context. This format must be used if the key is encrypted.

"KEYD0200 format" on page 51
Key parameters.

See "Key Description Formats" on page 51 for a description of these formats.

Cryptographic service provider
INPUT; CHAR(1)

The cryptographic service provider (CSP) that will perform the verify signature operation.

0 Any CSP.
The system will choose an appropriate CSP to perform the verify signature operation.

1 Software CSP.
The system will perform the verify signature operation using software. If the requested algorithm is not available in software, an error is returned.

2 Hardware CSP.
The system will perform the verify signature operation using cryptographic hardware. If the requested algorithm is not available in hardware, an error is returned. A specific cryptographic device can be specified using the cryptographic device name parameter. If the cryptographic device is not specified, the system will choose an appropriate one.

Cryptographic device name
INPUT; CHAR(10)

The name of a cryptographic device description.
This parameter is valid when the cryptographic service provider parameter specifies 2 (hardware CSP). Otherwise, this parameter must be blanks or the pointer to this parameter set to NULL.

Error code
I/O; CHAR(*)

The structure in which to return error information.
For the format of the structure, see Error Code Parameter.

Input Data Formats

For detailed descriptions of the table fields, see "**Input Data Formats Field Descriptions**" on page 50.

DATA0200 format

Offset			
Dec	**Hex**	**Type**	**Field**
These fields repeat.		PTR(SPP)	Input data pointer
		BINARY(4)	Input data length
		CHAR(12)	Reserved

Input Data Formats Field Descriptions

Input data length
The length of data to verify.

Input data pointer
A space pointer to the data to verify.

Reserved
Must be null (binary 0s).

Algorithm Description Formats

For detailed descriptions of the table fields, see "**Algorithm Description Formats Field Descriptions**."

ALGD0100 format

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	CHAR(8)	Algorithm context token
8	8	CHAR(1)	Final operation flag

ALGD0400 format

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	BINARY(4)	Public key cipher algorithm
4	4	CHAR(1)	PKA block format
5	5	CHAR(3)	Reserved
8	8	BINARY(4)	Signing hash algorithm

Algorithm Description Formats Field Descriptions

Algorithm context token
A token for an algorithm context. The algorithm context is created using the Create Algorithm Context (OPM, QC3CRTAX; ILE, Qc3CreateAlgorithmContext) API.

Final operation flag
The final processing indicator.

0 Continue.
The system will not perform final processing and the algorithm context will maintain the state of the operation. The algorithm context can be used on future calls to this API to continue the verify signature operation. The result of the signature verification will not be returned until the final operation flag is set on. The pointer to the signature parameter may be set to NULL because the signature is not used until the final operation flag is set on.

1 Final.
The system will perform final processing. The signature will be verified and the algorithm context will reset to its initial state. The algorithm context can then be used to begin a new cryptographic operation. When performing a final operation, the pointer to the input data parameter may be set to NULL.

PKA block format
The public key algorithm block format. Following are the valid values.

0 PKCS #1 block type 00
1 PKCS #1 block type 01
3 ISO 9796-1

5 ANSI X9.31
This format is only valid with signing hash algorithm 2 (SHA-1).

Public key cipher algorithm
The encryption algorithm. Following are the valid public key cipher algorithms.

50 RSA

Reserved
Must be null (binary 0s).

Signing hash algorithm
The hash algorithm. Following are the valid values for the signing hash algorithm.

1 MD5
2 SHA-1

Key Description Formats

For detailed descriptions of the table fields, see "**Key Description Formats Field Descriptions**."

KEYD0100 format

Offset		Type	Field
Dec	**Hex**		
0	0	CHAR(8)	Key context token

KEYD0200 format

Offset		Type	Field
Dec	**Hex**		
0	0	BINARY(4)	Key type
4	4	BINARY(4)	Key string length
8	8	CHAR(1)	Key format
9	9	CHAR(3)	Reserved
12	C	CHAR(*)	Key string

Key Description Formats Field Descriptions

Key context token
A token for a key context. The key context is created using the Create Key Context (OPM, QC3CRTKX; ILE, Qc3CreateKeyContext) API.

Key format
The format of the key string field. Following are the valid values.

1 BER string
The key is specified in BER encoded X.509 SubjectPublicKeyInfo format. For specifications of this format, refer to RFC 3280.

Key string
The key to use in the verify signature operation.

Key string length
Length of the key string specified in the key string field. The format of the key string is specified in the key format field.

Key type
The type of key. Following are the valid values.

50 RSA public

Reserved
Must be null (binary 0s).

Error Messages

Message ID	Error Message Text
CPF24B4 E	Severe error while addressing parameter list.
CPF3C1E E	Required parameter &1 omitted.
CPF3CF1 E	Error code parameter not valid.
CPF3CF2 E	Error(s) occurred during running of &1 API.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.
CPF9DC2 E	Key-encrypting algorithm context not compatible with key-encrypting key context.
CPF9DC6 E	Algorithm not valid for encrypting or decrypting a key.
CPF9DC9 E	The total length of data in the input data array is not valid.
CPF9DCC E	The length of area provided for signature is not valid.
CPF9DCE E	A data length is not valid.
CPF9DCF E	A data pointer is not valid.
CPF9DD0 E	Clear data format name not valid.
CPF9DD2 E	Algorithm description format name not valid.
CPF9DD3 E	Key description format name not valid.
CPF9DD5 E	Length of input data not valid.
CPF9DD6 E	Length of area provided for output data is too small.
CPF9DD7 E	The key-encrypting key context for the specified key is not valid or was previously destroyed.
CPF9DD8 E	The key-encrypting algorithm context for the specified key is not valid or was previously destroyed.
CPF9DDA E	Unexpected return code &1.
CPF9DDB E	The key string or Diffie-Hellman parameter string is not valid.
CPF9DDD E	The key string length is not valid.
CPF9DE0 E	Hash algorithm not valid.
CPF9DE3 E	Mode not valid.
CPF9DE5 E	PKA (public key algorithm) block format not valid.
CPF9DE7 E	Key type not valid.
CPF9DE9 E	Key format not valid.
CPF9DEC E	Cryptographic service provider not valid.
CPF9DED E	Final operation flag not valid.
CPF9DEE E	Reserved field not null.
CPF9DEF E	The signature verification failed.
CPF9DF0 E	Operation, algorithm, or mode not available on the requested CSP (cryptographic service provider).
CPF9DF1 E	The algorithm context token does not reference a valid algorithm context.
CPF9DF2 E	The algorithm context is not found or was previously destroyed.
CPF9DF3 E	Algorithm in algorithm context not valid for requested operation.
CPF9DF4 E	The key context token does not reference a valid key context.
CPF9DF5 E	The key context is not found or was previously destroyed.
CPF9DF7 E	Algorithm context not compatible with key context.
CPF9DF8 E	Cryptographic device name not valid.
CPF9DF9 E	Cryptographic device not found.
CPF9DFB E	Cryptographic service provider (CSP) conflicts with the key context CSP.
CPF9DFD E	Not authorized to device.
CPF9DFE E	Cryptographic device not available.

API introduced: V5R3

Top | "Cryptographic Services APIs," on page 1 | APIs by category

Key Generation APIs

Most cryptographic operations involve a mathematical formula (algorithm) and secret data (key). The Key Generation APIs allow you to generate random key values for both symmetric and asymmetric (PKA) algorithms.

The Key Generation APIs include:

- "Calculate Diffie-Hellman Secret Key (QC3CALDS, Qc3CalculateDHSecretKey)" on page 54 (QC3CALDS, Qc3CalculateDHSecretKey) calculates a Diffie-Hellman shared secret key.
- "Generate Diffie-Hellman Key Pair (QC3GENDK, Qc3GenDHKeyPair)" on page 56 (QC3GENDK, Qc3GenDHKeyPair) generates a Diffie-Hellman (D-H) private/public key pair needed for calculating a Diffie-Hellman shared secret key.
- "Generate Diffie-Hellman Parameters (QC3GENDP, Qc3GenDHParms)" on page 59 (QC3GENDP, Qc3GenDHParms) generates the parameters needed for generating a Diffie-Hellman key pair.
- "Generate PKA Key Pair (QC3GENPK, Qc3GenPKAKeyPair)" on page 61 (QC3GENPK, Qc3GenPKAKeyPair) generates a random PKA key pair.
- "Generate Symmetric Key (QC3GENSK, Qc3GenSymmetricKey)" on page 65 (QC3GENSK, Qc3GenSymmetricKey) generates a random key value that can be used with a symmetric cipher algorithm

Top | Cryptographic Services APIs | APIs by category

Calculate Diffie-Hellman Secret Key (QC3CALDS, Qc3CalculateDHSecretKey)

Required Parameter Group:	
1	D-H algorithm context token
Input	Char(8)
2	D-H public key
Input	Char(*)
3	Length of D-H public key
Input	Binary(4)
4	D-H secret key
Output	Char(*)
5	Length of area provided for D-H secret key
Input	Binary(4)
6	Length of D-H secret key returned
Output	Binary(4)
7	Error code
I/O	Char(*)
Service Program Name: QC3DH	
Default Public Authority: *USE	
Threadsafe: Yes	

Diffie-Hellman (D-H) is a public key algorithm used for producing a shared secret key. It is described in RFC 2631 and Public Key Cryptography Standard (PKCS) #3. To share a secret key between two parties, both parties calculate the shared secret key using their own private key and the other party's public key. To share a secret key with more than two parties, see the example below.

Authorities and Locks

Required API authority
*USE

Required device description authority
*USE

Required Parameter Group

D-H algorithm context token
INPUT; CHAR(8)

The token for the D-H algorithm context.
This must be the token for the algorithm context that was created using the "Generate Diffie-Hellman Key Pair (QC3GENDK, Qc3GenDHKeyPair)" on page 56. The D-H parameters and private key are contained in the context. Once the D-H secret key has been calculated, you should destroy the D-H algorithm context using the "Destroy Algorithm Context (QC3DESAX, Qc3DestroyAlgorithmContext)" on page 81.

D-H public key
INPUT; CHAR(*)

The other party's D-H public key.
This is the public key from the party with whom the secret key will be shared

Length of D-H public key
INPUT; BINARY(4)

The length of key specified in the D-H public key parameter.

D-H secret key
OUTPUT; CHAR(*)

The area to store the D-H secret key.
The entire output of the secret key may not be needed and the two parties must agree on which bytes of the secret value will be used.

Length of area provided for D-H secret key
INPUT; BINARY(4)

The length of the D-H secret key parameter in bytes.
The size of the secret key will be no greater than the key size. (See "Generate Diffie-Hellman Parameters (QC3GENDP, Qc3GenDHParms)" on page 59.) Because key size is normally specified in bits, divide that value by 8 and round up to obtain the length of area needed for the D-H secret key.

Length of D-H secret key returned
OUTPUT; BINARY(4)

The length of the D-H secret key returned in the D-H secret key parameter.
If the length of area provided is too small, an error will be generated and no data will be returned in the D-H secret key parameter.

Error code
I/O; CHAR(*)

The structure in which to return error information.
For the format of the structure, see Error Code Parameter.

Error Messages

Message ID	Error Message Text
CPF24B4 E	Severe error while addressing parameter list.
CPF3C1E E	Required parameter &1 omitted.
CPF3CF1 E	Error code parameter not valid.
CPF3CF2 E	Error(s) occurred during running of &1 API.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.
CPF9DCA E	Length of D-H (Diffie-Hellman) public key not valid.
CPF9DD6 E	Length of area provided for output data is too small.
CPF9DDA E	Unexpected return code &1.
CPF9DF1 E	The algorithm context token does not reference a valid algorithm context.
CPF9DF2 E	The algorithm context is not found or was previously destroyed.

Example of Three-Party Shared Secret Key Exchange

1. Beth uses Generate Diffie-Hellman Parameters and sends the output to Kathy and Terry.
2. Beth uses Generate Diffie-Hellman Key Pair to generate a private value (stored in a Diffie-Hellman algorithm context), and a public value B1, which she sends to Kathy.
3. Kathy uses Generate Diffie-Hellman Key Pair to generate a private value (stored in a Diffie-Hellman algorithm context), and a public value K1, which she sends to Terry.

4. Terry uses Generate Diffie-Hellman Key Pair to generate a private value (stored in a Diffie-Hellman algorithm context), and a public value T1, which he sends to Beth.
5. Beth specifies T1 on Calculate Diffie-Hellman Secret Key to create another public value B2, which she sends to Kathy.
6. Kathy specifies B1 on Calculate Diffie-Hellman Secret Key to create another public value K2, which she sends to Terry.
7. Terry specifies K1 on Calculate Diffie-Hellman Secret Key to create another public value T2, which he sends to Beth.
8. Beth specifies T2 on Calculate Diffie-Hellman Secret Key to create the shared secret key, S.
9. Kathy specifies B2 on Calculate Diffie-Hellman Secret Key to create the shared secret key, S.
10. Terry specifies K2 on Calculate Diffie-Hellman Secret Key to create the shared secret key, S.

API introduced: V5R3

Top | "Cryptographic Services APIs," on page 1 | APIs by category

Generate Diffie-Hellman Key Pair (QC3GENDK, Qc3GenDHKeyPair)

Required Parameter Group:

1	D-H parameters
Input	Char(*)
2	Length of D-H parameters
Input	Binary(4)
3	Cryptographic service provider
Input	Char(1)
4	Cryptographic device name
Input	Char(10)
5	D-H algorithm context token
Output	Char(8)
6	D-H public key
Output	Char(*)
7	Length of area provided for D-H public key
Input	Binary(4)
8	Length of D-H public key returned
Output	Binary(4)
9	Error code
I/O	Char(*)

Service Program Name: QC3DH

Default Public Authority: *USE

Threadsafe: Yes

Diffie-Hellman (D-H) is a public key algorithm used for producing a shared secret key. It is described in RFC 2631 and Public Key Cryptography Standard (PKCS) #3. The **Generate Diffie-Hellman Key Pair (OPM, QC3GENDK; ILE, Qc3GenDHKeyPair) API** generates a Diffie-Hellman (D-H) private/public key pair. The key pair is used to create a shared secret key using the "Calculate Diffie-Hellman Secret Key (QC3CALDS, Qc3CalculateDHSecretKey)" on page 54. The key pair can not be used for data encryption or signing.

Authorities and Locks

Required API authority
*USE

Required device description authority
*USE

Required Parameter Group

D-H parameters
INPUT; CHAR(*)

The ASN.1 BER encoded D-H parameters.
These parameters are obtained from the "Generate Diffie-Hellman Parameters (QC3GENDP, Qc3GenDHParms)" on page 59 or from another party.

Length of D-H parameters
INPUT; BINARY(4)

The length of the D-H parameters.

Cryptographic service provider
INPUT; CHAR(1)

The cryptographic service provider (CSP) that will perform the D-H operations (both generate D-H key pair and calucalte D-H secret key).

0 Any CSP.
The system will choose an appropriate CSP to perform the D-H operations.

1 Software CSP.
The system will perform the D-H operations using software. If the requested algorithm is not available in software, an error is returned.

2 Hardware CSP.
The system will perform the D-H operations using cryptographic hardware. If the requested algorithm is not available in hardware, an error is returned. A specific cryptographic device can be specified using the cryptographic device name parameter. If the cryptographic device is not specified, the system will choose an appropriate one.

Cryptographic device name
INPUT; CHAR(10)

The name of a cryptographic device description.
This parameter is valid when the cryptographic service provider parameter specifies 2 (hardware CSP). Otherwise, this parameter must be blanks or the pointer to this parameter set to NULL.

D-H algorithm context token
OUTPUT; CHAR(8)

The area to store the token for the created D-H algorithm context.
The D-H parameters and private key will be stored in the context upon completion of this operation. This token should be supplied on the "Calculate Diffie-Hellman Secret Key (QC3CALDS, Qc3CalculateDHSecretKey)" on page 54. Once the D-H secret key has been calculated, you should destroy the D-H algorithm context using the "Destroy Algorithm Context (QC3DESAX, Qc3DestroyAlgorithmContext)" on page 81.

D-H public key
OUTPUT; CHAR(*)

The area to store the D-H public key.
The D-H public key must be given to the party with whom the secret key will be shared.

Length of area provided for D-H public key
INPUT; BINARY(4)

The length of the D-H public key parameter in bytes.
The size of the public key will be no greater than the key size. (See "Generate Diffie-Hellman Parameters (QC3GENDP, Qc3GenDHParms)" on page 59.) Because key size is normally specified in bits, divide that value by 8 to obtain the length of area needed for the D-H public key.

Length of D-H public key returned
OUTPUT; BINARY(4)

The length of the generated D-H public key returned in the D-H public key parameter.
If the length of area provided is too small, an error will be generated and no data will be returned in the D-H public key parameter.

Error code
I/O; CHAR(*)

The structure in which to return error information.
For the format of the structure, see Error Code Parameter.

Error Messages

Message ID	Error Message Text
CPF24B4 E	Severe error while addressing parameter list.
CPF3C1E E	Required parameter &1 omitted.
CPF3CF1 E	Error code parameter not valid.
CPF3CF2 E	Error(s) occurred during running of &1 API.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.
CPF9DCB E	Length of D-H (Diffie-Hellman) parameters not valid.
CPF9DD6 E	Length of area provided for output data is too small.
CPF9DDA E	Unexpected return code &1.
CPF9DDB E	The key string or Diffie-Hellman parameter string is not valid.
CPF9DDC E	D-H (Diffie-Hellman) parameters not valid.
CPF9DEC E	Cryptographic service provider not valid.
CPF9DF8 E	Cryptographic device name not valid.
CPF9DF9 E	Cryptographic device not found.
CPF9DFD E	Not authorized to device.
CPF9DFE E	Cryptographic device not available.

API introduced: V5R3

Top | "Cryptographic Services APIs," on page 1 | APIs by category

Generate Diffie-Hellman Parameters (QC3GENDP, Qc3GenDHParms)

Required Parameter Group:	
1	Key size
Input	Binary(4)
2	Cryptographic service provider
Input	Char(1)
3	Cryptographic device name
Input	Char(10)
4	D-H parms
Output	Char(*)
5	Length of area provided for D-H parms
Input	Binary(4)
6	Length of D-H parms returned
Output	Binary(4)
7	Error code
I/O	Char(*)
Service Program Name: QC3DH	
Default Public Authority: *USE	
Threadsafe: Yes	

Diffie-Hellman (D-H) is a public key algorithm used for producing a shared secret key. It is described in RFC 2631 and Public Key Cryptography Standard (PKCS) #3. The output from the **Generate Diffie-Hellman Parameters (OPM, QC3GENDH; ILE, Qc3GenDHParms) API** is used in generating a D-H key pair (Generate Diffie-Hellman Key Pair (OPM, QC3GENDK; ILE, Qc3GenDHKeyPair) API). These parameters are not secret and must be given to the party (or parties) with whom a secret key will be shared. Alternatively, the D-H parameters may be supplied by another party.

Authorities and Locks

Required API authority
*USE

Required device description authority
*USE

Required Parameter Group

Key size
INPUT; BINARY(4)

The length of the modulus in bits.
The key size must be a multiple of 64 with a minimum size of 512 and a maximum size of 1024.

Cryptographic service provider
INPUT; CHAR(1)

The cryptographic service provider (CSP) that will perform the D-H operation.

1	Software CSP. The system will perform the D-H operation using software.

Cryptographic device name
INPUT; CHAR(10)

This parameter must be set to blanks or the pointer to this parameter set to NULL.

D-H parms
OUTPUT; CHAR(*)

The area to store the D-H parameters.
The generated D-H parameters will be returned in BER encoded PKCS #3 format. For specifications of this format, refer to RSA Security Inc. Public-Key Cryptography Standards. The D-H parameters are used in generating a Diffie-Hellman key pair and must be given to the party with whom the secret key will be shared. The generated parameters are not sensitive and need not be kept secret.

Length of area provided for D-H parms
INPUT; BINARY(4)

The length of the D-H parms parameter.
The maximum length needed (with a key size of 1024) is 288 bytes.

Length of D-H parms returned
OUTPUT; BINARY(4)

The length of the generated D-H parameters returned in the D-H parms parameter.
If the length of area provided is too small, an error will be generated and no data will be returned in the D-H parms parameter.

Error code
I/O; CHAR(*)

The structure in which to return error information.
For the format of the structure, see Error Code Parameter.

Error Messages

Message ID	Error Message Text
CPF24B4 E	Severe error while addressing parameter list.
CPF3C1E E	Required parameter &1 omitted.
CPF3CF1 E	Error code parameter not valid.
CPF3CF2 E	Error(s) occurred during running of &1 API.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.
CPF9DD6 E	Length of area provided for output data is too small.
CPF9DDA E	Unexpected return code &1.
CPF9DEA E	Key size not valid.
CPF9DEC E	Cryptographic service provider not valid.
CPF9DF8 E	Cryptographic device name not valid.

API introduced: V5R3

Top | Other APIs in this part | APIs by category

Generate PKA Key Pair (QC3GENPK, Qc3GenPKAKeyPair)

Required Parameter Group:

1	Key type
Input	Binary(4)
2	Key size
Input	Binary(4)
3	Public key exponent
Input	Binary(4)
4	Key format
Input	Char(1)
5	Key form
Input	Char(1)
6	Key-encrypting key context token
Input	Char(8)
7	Key-encrypting algorithm context token
Input	Char(8)
8	Cryptographic service provider
Input	Char(1)
9	Cryptographic device name
Input	Char(10)
10	Private key string
Output	Char(*)
11	Length of area provided for private key string
Input	Binary(4)
12	Length of private key string returned
Output	Binary(4)
13	Public Key string
Output	Char(*)
14	Length of area provided for public key string
Input	Binary(4)
15	Length of public key string returned
Output	Binary(4)
16	Error code
I/O	Char(*)

Service Program Name: QC3KEYGN

Default Public Authority: *USE

Threadsafe: Yes

The **Generate PKA Key Pair (OPM, QC3GENPK; ILE, Qc3GenPKAKeyPair) API** generates a random PKA key pair that can be used with the PKA cipher algorithm RSA.

Authorities and Locks

Required API authority
*USE

Required device description authority
*USE

Required Parameter Group

Key type
INPUT; BINARY(4)

The type of key.
Following are the valid values.

50 RSA

Key size
INPUT; BINARY(4)

The modulus length in bits.
The key size must be an even number in the range 512 - 2048.

Public key exponent
INPUT; BINARY(4)

To maximize performance, the public key exponent is limited to the following two values.

3 Or hex 00 00 00 03.
65,537 Or hex 00 01 00 01.

Key format
INPUT; CHAR(1)

The format in which to return the key.
Following are the valid values.

1 BER string. The private key is returned in BER encoded PKCS #8 format. For specifications of this format, refer to RSA Security Inc. Public-Key Cryptography Standards. The public key is returned in BER encoded X.509 SubjectPublicKeyInfo format. For specifications of this format, refer to RFC 3280.

Key form
INPUT; CHAR(1)

The form in which to return the private key string.

0 Clear.
The key string is returned in the clear.
1 Encrypted.
The private key string is returned encrypted. The key-encrypting key context token and key-encrypting algorithm context token parameters are used to encrypt the private key string before returning it.

Key-encrypting key context token
INPUT; CHAR(8)

The key context token for encrypting the private key string. If the key form parameter does not specify 1 (encrypted), this field must be blanks or the pointer to this parameter set to NULL.

Key-encrypting algorithm context token
INPUT; CHAR(8)

The algorithm context token for encrypting the private key string. If the key form parameter does not specify 1 (encrypted), this field must be blanks or the pointer to this parameter set to NULL.

Cryptographic service provider
INPUT; CHAR(1)

The cryptographic service provider (CSP) that will perform the key generate operation.

1	Software CSP. The system will perform the PKA key pair generation using software.

Cryptographic device name
INPUT; CHAR(10)

This parameter must be set to blanks or the pointer to this parameter set to NULL.

Private key string
OUTPUT; CHAR(*)

The area to store the generated private key string or the pointer to this parameter set to NULL.

Length of area provided for the private key string
INPUT; BINARY(4)

The length of the private key string parameter. At most, the generated private key string will be 1504 bytes.

Length of private key string returned
OUTPUT; BINARY(4)

The length of the generated private key string returned in the private key string parameter.
If the length of area provided is too small, an error will be generated and no data will be returned in the private key string parameter.

Public key string
OUTPUT; CHAR(*)

The area to store the public key string.

Length of area provided for the public key string
INPUT; BINARY(4)

The length of the public key string parameter. At most, the public key string will be 512 bytes.

Length of public key string returned
OUTPUT; BINARY(4)

The length of the public key string returned in the public key string parameter.
If the length of area provided is too small, an error will be generated and no data will be returned in the public key string parameter.

Error code
I/O; CHAR(*)

The structure in which to return error information.
For the format of the structure, see Error Code Parameter.

Error Messages

Message ID	Error Message Text
CPF24B4 E	Severe error while addressing parameter list.

Message ID	Error Message Text
CPF3C1E E	Required parameter &1 omitted.
CPF3CF1 E	Error code parameter not valid.
CPF3CF2 E	Error(s) occurred during running of &1 API.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.
CPF9DC2 E	Key-encrypting algorithm context not compatible with key-encrypting key context.
CPF9DC6 E	Algorithm not valid for encrypting or decrypting a key.
CPF9DD6 E	Length of area provided for output data is too small.
CPF9DDA E	Unexpected return code &1.
CPF9DE7 E	Key type not valid.
CPF9DE8 E	Key form not valid.
CPF9DE9 E	Key format not valid.
CPF9DEA E	Key size not valid.
CPF9DEB E	Public key exponent not valid.
CPF9DEC E	Cryptographic service provider not valid.
CPF9DF0 E	Operation, algorithm, or mode not available on the requested CSP (cryptographic service provider).
CPF9DF1 E	The algorithm context token does not reference a valid algorithm context.
CPF9DF2 E	The algorithm context is not found or was previously destroyed.
CPF9DF3 E	Algorithm in algorithm context not valid for requested operation.
CPF9DF4 E	The key context token does not reference a valid key context.
CPF9DF5 E	The key context is not found or was previously destroyed.
CPF9DF6 E	Key can not be encrypted.
CPF9DF7 E	Algorithm context not compatible with key context.
CPF9DF8 E	Cryptographic device name not valid.
CPF9DF9 E	Cryptographic device not found.
CPF9DFA E	Multiple-block encryption not valid with the requested mode.
CPF9DFD E	Not authorized to device.
CPF9DFE E	Cryptographic device not available.
CPF9DFF E	Request not allowed by cryptographic attributes.

API introduced: V5R3

Top | “Cryptographic Services APIs,” on page 1 | APIs by category

Generate Symmetric Key (QC3GENSK, Qc3GenSymmetricKey)

Required Parameter Group:	
1	Key type
Input	Binary(4)
2	Key size
Input	Binary(4)
3	Key format
Input	Char(1)
4	Key form
Input	Char(1)
5	Key-encrypting key context token
Input	Char(8)
6	Key-encrypting algorithm context token
Input	Char(8)
7	Cryptographic service provider
Input	Char(1)
8	Cryptographic device name
Input	Char(10)
9	Key string
Output	Char(*)
10	Length of area provided for key string
Input	Binary(4)
11	Length of key string returned
Output	Binary(4)
12	Error code
I/O	Char(*)
Service Program Name: QC3KEYGN	
Default Public Authority: *USE	
Threadsafe: Yes	

The **Generate Symmetric Key (OPM, QC3GENSK; ILE, Qc3GenSymmetricKey) API** generates a random key value that can be used with symmetric cipher algorithms DES, Triple DES, AES, RC2, and RC4-compatible.

Authorities and Locks

Required API authority
*USE

Required device description authority
*USE

Required Parameter Group

Key type
INPUT; BINARY(4)

The type of key.
Following are the valid values.

20 DES
Only 7 bits of each byte are used as the actual key. The rightmost bit of each byte will be set to odd parity because some cryptographic service providers require that a DES key have odd parity in every byte.
The key size parameter must specify 8.

21 Triple DES
Only 7 bits of each byte are used as the actual key. The rightmost bit of each byte will be set to odd parity because some cryptographic service providers require that a DES key have odd parity in every byte.
The key size can be 8, 16, or 24. Triple DES operates on an encryption block by doing a DES encrypt, followed by a DES decrypt, and then another DES encrypt. Therefore, it actually uses three 8-byte DES keys. If the key is 24 bytes in length, the first 8 bytes are used for key 1, the second 8 bytes for key 2, and the third 8 bytes for key 3. If the key is 16 bytes in length, the first 8 bytes are used for key 1 and key 3, and the second 8 bytes for key 2. If the key is only 8 bytes in length, it will be used for all 3 keys (essentially making the operation equivalent to a single DES operation).

22 AES
The key size can be 16, 24, or 32.
AES keys are supported only by the software CSP.

23 RC2
The key size can be 1 - 128.
RC2 keys are supported only by the software CSP.

30 RC4-compatible
The key size can be 1 - 256.
RC4-compatible keys are supported only by the software CSP. Because of the nature of the RC4-compatible operation, using the same key for more than one message will severely compromise security.

Key size
INPUT; BINARY(4)

The length of key to generate in bytes.
Refer to the key type parameter for restrictions.

Key format
INPUT; CHAR(1)

The format in which to return the key.
Following are the valid values.

0 Binary string.
The key is returned as a binary value.

Key form
INPUT; CHAR(1)

The form in which to return the key.

0 Clear.
The key string is returned in the clear.

1 Encrypted.
The key string is returned encrypted. The key-encrypting key context token and key-encrypting algorithm context token parameters are used to encrypt the key string before returning it.

Key-encrypting key context token
INPUT; CHAR(8)

The key context token for encrypting the key string. If the key form parameter does not specify 1 (encrypted), this field must be blanks or the pointer to this parameter set to NULL.

Key-encrypting algorithm context token
INPUT; CHAR(8)

The algorithm context token for encrypting the key string. If the key form parameter does not specify 1 (encrypted), this field must be blanks or the pointer to this parameter set to NULL.

Cryptographic service provider
INPUT; CHAR(1)

The cryptographic service provider (CSP) that will perform the key generate operation.

0 Any CSP.
The system will choose an appropriate CSP to perform the key generate operation.

1 Software CSP.
The system will perform the key generate operation using software. If the requested key type or form is not available in software, an error is returned.

2 Hardware CSP.
The system will perform the key generate operation using cryptographic hardware. If the requested key type or form is not available in hardware, an error is returned. A specific cryptographic device can be specified using the cryptographic device name parameter. If the cryptographic device is not specified, the system will choose an appropriate one.

Cryptographic device name
INPUT; CHAR(10)

The name of a cryptographic device description.
This parameter is valid when the cryptographic service provider parameter specifies 2 (hardware CSP). Otherwise, this parameter must be blanks or the pointer to this parameter set to NULL.

Key string
OUTPUT; CHAR(*)

The area to store the generated key string.

Length of area provided for key string
INPUT; BINARY(4)

The length of the key string parameter.
The length of the generated key string will be the length specified in the key size parameter. If the key form specifies 1 (encrypted), you must allow room for padding the encrypted key string to the next block length multiple. (e.g. Add an additional 8 bytes for DES.) For more information on block length, refer to the Create Algorithm Context (OPM, QC3CRTAX; ILE, Qc3CreateAlgorithmContext) API.

Length of key string returned
OUTPUT; BINARY(4)

The length of the key string returned in the key string parameter.
If the length of area provided for the key string is too small, an error will be generated and no data will be returned in the key string parameter.

Error code
I/O; CHAR(*)

The structure in which to return error information.
For the format of the structure, see Error Code Parameter.

Error Messages

Message ID	Error Message Text
CPF24B4 E	Severe error while addressing parameter list.
CPF3C1E E	Required parameter &1 omitted.
CPF3CF1 E	Error code parameter not valid.
CPF3CF2 E	Error(s) occurred during running of &1 API.

Message ID	Error Message Text
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.
CPF9DC2 E	Key-encrypting algorithm context not compatible with key-encrypting key context.
CPF9DC6 E	Algorithm not valid for encrypting or decrypting a key.
CPF9DD6 E	Length of area provided for output data is too small.
CPF9DD7 E	The key-encrypting key context for the specified key is not valid or was previously destroyed.
CPF9DD8 E	The key-encrypting algorithm context for the specified key is not valid or was previously destroyed.
CPF9DDA E	Unexpected return code &1.
CPF9DDB E	The key string or Diffie-Hellman parameter string is not valid.
CPF9DDD E	The key string length is not valid.
CPF9DE7 E	Key type not valid.
CPF9DE8 E	Key form not valid.
CPF9DE9 E	Key format not valid.
CPF9DEA E	Key size not valid.
CPF9DEC E	Cryptographic service provider not valid.
CPF9DF0 E	Operation, algorithm, or mode not available on the requested CSP (cryptographic service provider).
CPF9DF1 E	The algorithm context token does not reference a valid algorithm context.
CPF9DF2 E	The algorithm context is not found or was previously destroyed.
CPF9DF3 E	Algorithm in algorithm context not valid for requested operation.
CPF9DF4 E	The key context token does not reference a valid key context.
CPF9DF5 E	The key context is not found or was previously destroyed.
CPF9DF6 E	Key can not be encrypted.
CPF9DF7 E	Algorithm context not compatible with key context.
CPF9DF8 E	Cryptographic device name not valid.
CPF9DF9 E	Cryptographic device not found.
CPF9DFB E	Cryptographic service provider (CSP) conflicts with the key context CSP.
CPF9DFC E	The key-encrypting algorithm or key context token is not valid.
CPF9DFD E	Not authorized to device.
CPF9DFE E	Cryptographic device not available.
CPF9DFF E	Request not allowed by cryptographic attributes.

API introduced: V5R3

Top | "Cryptographic Services APIs," on page 1 | APIs by category

Pseudorandom Number Generation APIs

The Pseudorandom Number Generation APIs allow you to generate pseudorandom values that are statistically random and unpredictable (cryptographically secure).

The Pseudorandom Number Generation APIs include:

- "Add Seed for Pseudorandom Number Generator (QC3ADDSD, Qc3AddPRNGSeed) API" on page 69 (QC3ADDSD, Qc3AddPRNGSeed) allows the user to add seed into the server"s pseudorandom number generator system seed digest.
- "Generate Pseudorandom Numbers (QC3GENRN,Qc3GenPRNs) API" on page 70 (QC3ADDSD, Qc3GenPRNs) generates a pseudorandom binary stream.

Top | Cryptographic Services APIs | APIs by category

Add Seed for Pseudorandom Number Generator (QC3ADDSD, Qc3AddPRNGSeed) API

Required Parameter Group:	
1	Seed data
Input	Char(*)
2	Seed data length
Input	Binary(4)
3	Error Code
I/O	Char(*)
Service Program Name: QC3PRNG	
Default Public Authority: *USE	
Threadsafe: Yes	

The Add Seed for Pseudorandom Number Generator (OPM, QC3ADDSD; ILE, Qc3AddPRNGSeed) API allows the user to add seed into the server's pseudorandom number generator system seed digest.

The pseudorandom number generator is composed of two parts: pseudorandom number generation and seed management. Pseudorandom number generation is performed using the FIPS 186-1 algorithm. (See the Generate Pseudorandom Numbers (Qc3GenPRNs) API.) Cryptographically-secure pseudorandom numbers rely on good seed. The FIPS 186-1 key and seed values are obtained from the system seed digest. The server automatically generates seed using data collected from system information or by using the random number generator function on a cryptographic coprocessor, such as a 4758, if one is available. System-generated seed can never be truly unpredictable. If a cryptographic coprocessor is not available, you can use this API to add your own random seed to the system seed digest. This should be done as soon as possible any time the Licensed Internal Code is installed.

Authorities and Locks

All object (*ALLOBJ) special authority is needed to use this API.

User Profile Authority
*ALLOBJ

Required Parameter Group

Seed data
INPUT; CHAR(*)

The input seed data for the system seed digest.

It is important that the seed data be unpredictable and have as much entropy as possible. Entropy is the minimum number of bits needed to represent the information contained in some data. For seeding purposes, entropy is a measure of the amount of uncertainty or unpredictability of the seed. The system seed digest holds a maximum of 160 bits of entropy. You should add at least that much entropy to refresh the system seed digest totally. Possible sources of seed data are coin flipping, keystroke or mouse timings, or a noise source such as the one available on the 4758 Cryptographic Coprocessor.

Seed data length
INPUT; BINARY(4)

The length of the seed data, in bytes. If this length is 0, no seed data is added.

Error code
I/O; CHAR(*)

The structure in which to return error information. For the format of the structure, see Error Code Parameter.

Error Messages

Message ID	Error Message Text
CPF222E E	*ALLOBJ special authority is required.
CPF3C17 E	Error occurred with input data parameter.
CPF3CF1 E	Error code parameter not valid.

API introduced: V5R1

Top | Miscellaneous APIs | APIs by category

Generate Pseudorandom Numbers (QC3GENRN,Qc3GenPRNs) API

Required Parameter Group:	
1	PRN data
Output	Char(*)
2	PRN data length
Input	Binary(4)
3	PRN type
Input	Char(1)
4	PRN Parity
Input	Char(1)
5	Error code
I/O	Char(*)

Service Program Name: QC3PRNG

Default Public Authority: *USE

Threadsafe: Yes

The Generate Pseudorandom Numbers (OPM, QC3GENRN; ILE, Qc3GenPRNs) API generates a pseudorandom binary stream.

The pseudorandom number generator is composed of two parts: pseudorandom number generation and seed management. Pseudorandom number generation is performed using the FIPS 186-1 algorithm. Cryptographically-secure pseudorandom numbers rely on good seed. The FIPS 186-1 key and seed values are obtained from the system seed digest. The server automatically generates seed using data collected from system information or by using the random number generator function on a cryptographic

coprocessor, such as a 4758, if one is available. System-generated seed can never be truly unpredictable. If a cryptographic coprocessor is not available, you can use the Add Seed for PRNG (Qc3AddPRNGSeed) API to add your own random seed to the system seed digest. This should be done as soon as possible any time the Licensed Internal Code is installed.

Authorities and Locks

None.

Required Parameter Group

PRN data
OUTPUT; CHAR(*)

The generated pseudorandom binary stream.

PRN data length
INPUT; BINARY(4)

The number of pseudorandom number bytes to return in the PRN data parameter. If 0 is specified, no pseudorandom numbers are returned.

PRN type
INPUT; CHAR(1)

The API can generate a real pseudorandom binary stream or a test binary stream.

The FIPS 186-1 algorithm obtains the inital key and seed values from the system seed digest when generating a real pseudorandom binary stream. When generating a test binary stream, the algorithm uses preset values for the key and seed. Valid values are:

0 Generate real pseudorandom numbers.
1 Generate test pseudorandom numbers.

PRN Parity
INPUT; CHAR(1)

The API sets each byte of the pseudorandom number binary stream to the specified parity by altering the low order bit in each byte as necessary. Valid values are:

0 Do not set parity.
1 Set each byte to odd parity.
2 Set each byte to even parity.

Error code
I/O; CHAR(*)

The structure in which to return error information. For the format of the structure, see Error Code Parameter.

Error Messages

Message ID	Error Message Text
CPF3C19 E	Error occurred with receiver variable specified.
CPF3CF1 E	Error code parameter not valid.
CPFBAF1 E	PRN type not valid.
CPFBAF2 E	Parity not valid.
CPFBAF3 E	The system seed digest is not ready.

API introduced: V5R1

Top | Miscellaneous APIs | APIs by category

Cryptographic Context APIs

The Cryptographic Context APIs are used to temporarily store the key and algorithm parameters for cryptographic operations.

The Cryptographic Context APIs include:

- "Create Algorithm Context (QC3CRTAX, Qc3CreateAlgorithmContext)" (QC3CRTAX, Qc3CreateAlgorithmContext) creates a temporary area for holding the algorithm parameters and the state of the cryptographic operation.
- "Create Key Context (QC3CRTKX, Qc3CreateKeyContext)" on page 78 (QC3CRTKX, Qc3CreateKeyContext) creates a temporary area for holding a cryptographic key.
- "Destroy Algorithm Context (QC3DESAX, Qc3DestroyAlgorithmContext)" on page 81 (QC3DESAX, Qc3DestroyAlgorithmContext) destroys the algorithm context created with the Create Algorithm Context (OPM: QC3CRTAX; ILE: Qc3CreateAlgorithmContext) API.
- "Destroy Key Context (QC3DESKX, Qc3DestroyKeyContext)" on page 82 (QC3DESKX, Qc3DestroyKeyContext) destroys the key context created with the Create Key Context (OPM: QC3CRTKX; ILE: Qc3CreateKeyContext) API.

Top | Cryptographic Services APIs | APIs by category

Create Algorithm Context (QC3CRTAX, Qc3CreateAlgorithmContext)

Required Parameter Group:

1	Algorithm description
Input	Char(*)
2	Algorithm description format name
Input	Char(8)
3	Algorithm context token
Output	Char(8)
4	Error code
I/O	Char(*)

Service Program Name: QC3CTX

Default Public Authority: *USE

Threadsafe: Yes

The **Create Algorithm Context (OPM, QC3CRTAX; ILE, Qc3CreateAlgorithmContext) API** creates a temporary area for holding the algorithm parameters and the state of the cryptographic operation. The API returns a token which can be used on subsequent cryptographic APIs. The algorithm context token can be used to extend a cryptographic operation over multiple calls. The algorithm context can not be shared between jobs. It should be destroyed using the "Destroy Algorithm Context (QC3DESAX, Qc3DestroyAlgorithmContext)" on page 81. If not explicitly destroyed, the algorithm context will be destroyed at job end.

Authorities and Locks

Required API authority

*USE

Required Parameter Group

Algorithm description

INPUT; CHAR(*)

The algorithm and associated parameters.
The format of the algorithm description is specified in the algorithm description format name parameter.

Algorithm description format name

INPUT; CHAR(8)

The format of the algorithm description.
The possible format names follow.

"ALGD0200 format"	Block cipher algorithm (DES, Triple DES, AES, and RC2).
"ALGD0300 format" on page 74	Stream cipher algorithm (RC4-compatible).
"ALGD0400 format" on page 74	Public key algorithm (RSA).
"ALGD0500 format" on page 74	Hash algorithm (MD5, SHA-1, SHA-256, SHA-384, SHA-512).

See "Algorithm Description Formats" for a description of these formats.

Algorithm context token

OUTPUT; CHAR(8)

The area to store the token for the created algorithm context.
Each token will contain an authentication value. If the token is used on a subsequent API but with an incorrect authentication value, the user will be subjected to a 10 second penalty wait. For each authentication error in that job, the penalty wait will increase 10 seconds up to a maximum of 10 minutes.

Error code

I/O; CHAR(*)

The structure in which to return error information.
For the format of the structure, see Error Code Parameter.

Algorithm Description Formats

For detailed descriptions of the fields in the tables, see "**Algorithm Description Formats Field Descriptions**" on page 74.

ALGD0200 format

Offset		Type	Field
Dec	**Hex**		
0	0	BINARY(4)	Block cipher algorithm

Offset		Type	Field
Dec	Hex		
4	4	BINARY(4)	Block length
8	8	CHAR(1)	Mode
9	9	CHAR(1)	Pad option
10	A	CHAR(1)	Pad character
11	B	CHAR(1)	Reserved
12	C	BINARY(4)	MAC length
16	10	BINARY(4)	Effective key size
20	14	CHAR(32)	Initialization vector

ALGD0300 format

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Stream cipher algorithm

ALGD0400 format

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Public key algorithm
4	4	CHAR(1)	PKA block format
5	5	CHAR(3)	Reserved
8	8	BINARY(4)	Signing hash algorithm

ALGD0500 format

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Hash algorithm

Algorithm Description Formats Field Descriptions

Block cipher algorithm

Following are the valid block cipher algorithms:

20 DES
Documented in FIPS 46-3. DES is no longer considered secure enough for today's fast computers. It should be used for compatibility purposes only.

21 Triple DES
Documented in FIPS 46-3.

22 AES
Documented in FIPS 197.

23 RC2
Documented in RFC 2268.

Block length

The algorithm block length. For DES, Triple DES, and RC2, this field must specify 8. The valid block length values for AES are 16, 24, and 32. Note: 16 is the only block length specified in the AES standard, FIPS-197.

Effective key size

For RC2, the number of key bits to use in the cipher operation. Valid values are from 1 to 1024. If RC2 is not specifed for the block cipher algorithm, this field must be set to 0.

Hash algorithm

Following are the valid hash algorithms:

1 MD5
Documented in RFC 1321.
2 SHA-1
Documented in FIPS 180-2.
3 SHA-256
Documented in FIPS 180-2.
4 SHA-384
Documented in FIPS 180-2.
5 SHA-512
Documented in FIPS 180-2.

Initialization vector

The initialization vector (IV). Refer to the mode standards for an explanation of its use. For DES, Triple DES, and RC2, the first 8 bytes are used as the IV. For AES, the length of IV used is that specified by block length. The IV need not be secret, but it should be unique for each message. If not unique, it may compromise security. The IV can be any value. To obtain a good random IV value, use the "Generate Pseudorandom Numbers (QC3GENRN,Qc3GenPRNs) API" on page 70. An IV is not used for mode ECB, and must be set to NULL (hex 0's).

MAC length

The message authentication code length. This field is used only by the "Calculate MAC (QC3CALMA, Qc3CalculateMAC)" on page 34. MAC length can not exceed the block length value. When calculating a MAC, the leftmost MAC length bytes from the last block of encrypted data are returned as the MAC.

Mode

The mode of operation. Information on modes can be found in FIPS PUB 81 and ANSI X9.52. Following are the valid modes:

0 ECB
1 CBC
2 OFB. Not valid with AES or RC2.
3 CFB 1-bit. Not valid with AES or RC2.
4 CFB 8-bit. Not valid with AES or RC2.
5 CFB 64-bit. Not valid with AES or RC2.

Pad character

The pad character for pad option 1. Using hex 00 as the pad character is equivalent to ANSI X9.23 padding.

Pad option

Padding, if requested, is performed at the end of the operation. Be sure the area provided for the encrypted data is large enough to include the pad characters. The data will be padded up to the next block length byte multiple. For example, when using DES and and total data to encrypt is 20, the text is

padded to 24. The last byte is filled with a 1-byte binary counter containing the number of pad characters used. The preceeding pad characters are filled as specified by this field. Padding is not performed for modes CFB 1-bit and CFB 8-bit. In these cases, the pad option must be set to 0. Following are the valid pad options.

0 No padding is performed.
1 Use the character specified in the pad character field for padding.
2 The pad counter is used as the pad character. This is equivalent to PKCS #5 padding.

PKA block format

The public key algorithm block format. Following are the valid values:

0 PKCS #1 block type 00
1 PKCS #1 block type 01
2 PKCS #1 block type 02
This format is allowed on encryption and decryption operations only.
3 ISO 9796-1
This format is allowed on calculate signature and verify signature operations only. Because of security weaknesses, this format should be used for compatibility purposes only.
4 Zero pad
This format is allowed on encryption and decryption operations only. The clear data is placed in the low-order bit positions of a string of the same bit-length as the key modulus. All leading bits are set to zero.
5 ANSI X9.31
This format is allowed on calculate signature and verify signature operations only.

Public key algorithm

Following are the valid public key algorithms:

50 RSA
Documented in Public-Key Cryptography Standard (PKCS) #1.

Reserved

Must be null (binary 0s).

Signing hash algorithm

The hash algorithm for a sign or verify operation. Following are the valid values for the signing hash algorithm:

0 This algorithm context will not be used in a sign or verify operation.
1 MD5
Documented in RFC 1321.
2 SHA-1
Documented in FIPS 180-2.

Stream cipher algorithm

Following are the valid stream cipher algorithms:

30 RC4-compatible

Standards Resources

- FIPS publications are available from NIST Computer Security Resource Center at http://csrc.nist.gov/ .
- RFC publications are available from IETF at http://www.ietf.org/ .
- PKCS publications are available from RSA Security Inc. web pages.
- ANSI and ISO publications are available from ANSI Online at http://www.ansi.org/ .
- ISO publications are available from ISO Online at http://www.iso.ch/ .

Error Messages

Message ID	Error Message Text
CPF24B4 E	Severe error while addressing parameter list.
CPF3C1E E	Required parameter &1 omitted.
CPF3CF1 E	Error code parameter not valid.
CPF3CF2 E	Error(s) occurred during running of &1 API.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.
CPF9DD2 E	Algorithm description format name not valid.
CPF9DD9 E	Effective key size not valid.
CPF9DDA E	Unexpected return code &1.
CPF9DDE E	Cipher algorithm not valid.
CPF9DDF E	Block length not valid.
CPF9DE0 E	Hash algorithm not valid.
CPF9DE1 E	Initialization vector not valid.
CPF9DE2 E	MAC (message authentication code) length not valid.
CPF9DE3 E	Mode not valid.
CPF9DE4 E	Pad option not valid.
CPF9DE5 E	PKA (public key algorithm) block format not valid.
CPF9DE6 E	Public key algorithm not valid.
CPF9DEE E	Reserved field not null.

API introduced: V5R3

Top | "Cryptographic Services APIs," on page 1 | APIs by category

Create Key Context (QC3CRTKX, Qc3CreateKeyContext)

Required Parameter Group:	
1	Key string
Input	Char(*)
2	Length of key string
Input	Binary(4)
3	Key format
Input	Char(1)
4	Key type
Input	Binary(4)
5	Key form
Input	Char(1)
6	Key-encrypting key context token
Input	Char(8)
7	Key-encrypting algorithm context token
Input	Char(8)
8	Key context token
Output	Char(8)
9	Error code
I/O	Char(*)
Service Program Name: QC3CTX	
Default Public Authority: *USE	
Threadsafe: Yes	

The **Create Key Context (OPM, QC3CRTKX; ILE, Qc3CreateKeyContext) API** creates a temporary area for holding a cryptographic key. The API returns a token which can be used on subsequent cryptographic APIs when specifying a key. The key context can not be shared between jobs. It should be destroyed using the "Destroy Key Context (QC3DESKX, Qc3DestroyKeyContext)" on page 82. If the key context is not destroyed before relinquishing control, it could be used by other users of the job. If not explicitly destroyed, the key context will be destroyed at job end.

Authorities and Locks

Required API authority
*USE

Required Parameter Group

Key string
INPUT; CHAR(*)

The key. The format of the key string is specified in the key format parameter.

Length of key string
INPUT; BINARY(4)

Length of the key string specified in the key string parameter.
See the key type parameter for restrictions on key length.

Key format

INPUT; CHAR(1)

Format of the key string parameter.
Following are the valid values.

0 Binary string. The key is specified as a binary value. To obtain a good random key value, use the "Generate Symmetric Key (QC3GENSK, Qc3GenSymmetricKey)" on page 65, or "Generate Pseudorandom Numbers (QC3GENRN,Qc3GenPRNs) API" on page 70 API.

1 BER string. If the key type field specifies 50 (RSA public), the key must be specified in BER encoded X.509 SubjectPublicKeyInfo format. For specifications of this format, refer to RFC 3280. If the key type field specifies 51 (RSA private), the key must be specified in BER encoded PKCS #8 format. For specifications of this format, refer to RSA Security Inc. Public-Key Cryptography Standards. To generate a PKA key pair, use the "Generate PKA Key Pair (QC3GENPK, Qc3GenPKAKeyPair)" on page 61.

Key type

INPUT; BINARY(4)

The type of key.
Following are the valid values.

1 MD5
The key format must be 0. An MD5 key is used for HMAC (hash message authentication code) operations. The minimum length for an MD5 HMAC key is 16 bytes. A key longer than 16 bytes does not significantly increase the function strength unless the randomness of the key is considered weak. A key longer than 64 bytes will be hashed before it is used.

2 SHA-1
The key format must be 0. An SHA-1 key is used for HMAC (hash message authentication code) operations. The minimum length for an SHA-1 HMAC key is 20 bytes. A key longer than 20 bytes does not significantly increase the function strength unless the randomness of the key is considered weak. A key longer than 64 bytes will be hashed before it is used.

20 DES
The key format must be 0. It must be 8 bytes in length. Only 7 bits of each byte are used as the actual key. The rightmost bit of each byte is used to set parity. Some cryptographic service providers require that a DES key have odd parity in every byte. Others ignore parity.

21 Triple DES
The key format must be 0. It must be 8, 16, or 24 bytes in length. Triple DES operates on an encryption block by doing a DES encrypt, followed by a DES decrypt, and then another DES encrypt. Therefore, it actually uses three 8-byte DES keys. If 24 bytes are supplied in the key string, the first 8 bytes are used for key 1, the second 8 bytes for key 2, and the third 8 bytes for key 3. If 16 bytes are supplied, the first 8 bytes are used for key 1 and key 3, and the second 8 bytes for key 2. If only 8 bytes are supplied, it will be used for all 3 keys (essentially making the operation equivalent to a single DES operation). Only 7 bits of each byte are used as the actual key. The rightmost bit of each byte is used to set parity. Some cryptographic service providers require that a Triple DES key have odd parity in every byte. Others ignore parity.

22 AES
The key format must be 0. It must be 16, 24, or 32 bytes in length.

23 RC2
The key format must be 0. It must be from 1 to 128 bytes in length.

30 RC4-compatible
The key format must be 0. It must be from 1 to 256 bytes in length. Because of the nature of the RC4-compatible operation, using the same key for more than one message will severely compromise security.

50 RSA public
The key format must be 1.

51 RSA private
The key format must be 1.

Key form

INPUT; CHAR(1)

An indicator specifying if the key string parameter is in encrypted form.

0 Clear.
The key string is not encrypted.

1 Encrypted.
The key string is encrypted. The key-encrypting key context token and key-encrypting algorithm context token parameters are used to decrypt the key string when a cryptographic operation is performed.

Key-encrypting key context token
INPUT; CHAR(8)

The key context token specifying the key for decrypting the key string parameter. If the key string parameter is not encrypted (key form parameter is 0), this parameter must be set to blanks or the pointer to this parameter set to NULL.

Key-encrypting algorithm context token
INPUT; CHAR(8)

The algorithm context token specifying the algorithm for decrypting the key string parameter. If the key string parameter is not encrypted (key form parameter is 0), this parameter must be set to blanks or the pointer to this parameter set to NULL.

Key context token
OUTPUT; CHAR(8)

The area to store the token for the created key context.
Each token will contain an authentication value. If the token is used on a subsequent API but with an incorrect authentication value, the user will be subjected to a 10 second penalty wait. For each authentication error in that job, the penalty wait will increase 10 seconds up to a maximum of 10 minutes.

Error code
I/O; CHAR(*)

The structure in which to return error information.
For the format of the structure, see Error Code Parameter.

Error Messages

Message ID	Error Message Text
CPF24B4 E	Severe error while addressing parameter list.
CPF3C1E E	Required parameter &1 omitted.
CPF3CF2 E	Error(s) occurred during running of &1 API.
CPF3CF1 E	Error code parameter not valid.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.
CPF9DDA E	Unexpected return code &1.
CPF9DDD E	The key string length is not valid.
CPF9DE7 E	Key type not valid.
CPF9DE8 E	Key form not valid.
CPF9DE9 E	Key format not valid.
CPF9DF1 E	The algorithm context token does not reference a valid algorithm context.
CPF9DF2 E	The algorithm context is not found or was previously destroyed.
CPF9DF3 E	Algorithm in algorithm context not valid for requested operation.
CPF9DF4 E	The key context token does not reference a valid key context.
CPF9DF5 E	The key context is not found or was previously destroyed.
CPF9DF7 E	Algorithm context not compatible with key context.
CPF9DFC E	The key-encrypting algorithm or key context token is not valid.

«
API introduced: V5R3

Top | "Cryptographic Services APIs," on page 1 | APIs by category

Destroy Algorithm Context (QC3DESAX, Qc3DestroyAlgorithmContext)

Required Parameter Group:	
1	Algorithm context token
Input	Char(8)
2	Error code
I/O	Char(*)

Service Program Name: QC3CTX

Default Public Authority: *USE

Threadsafe: Yes

The **Destroy Algorithm Context (OPM, QC3DESAX; ILE, Qc3DestroyAlgorithmContext) API** destroys an algorithm context created by the "Create Algorithm Context (QC3CRTAX, Qc3CreateAlgorithmContext)" on page 72.

Authorities and Locks

Required API authority
*USE

Required Parameter Group

Algorithm context token
INPUT; CHAR(8)

The token of the algorithm context to destroy.

Error code
I/O; CHAR(*)

The structure in which to return error information.
For the format of the structure, see Error Code Parameter.

Error Messages

Message ID	Error Message Text
CPF3C1E E	Required parameter &1 omitted.
CPF3CF1 E	Error code parameter not valid.
CPF3CF2 E	Error(s) occurred during running of &1 API.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.
CPF9DDA E	Unexpected return code &1.
CPF9DF1 E	The algorithm context token does not reference a valid algorithm context.
CPF9DF2 E	The algorithm context is not found or was previously destroyed.

API introduced: V5R3

Top | "Cryptographic Services APIs," on page 1 | APIs by category

Destroy Key Context (QC3DESKX, Qc3DestroyKeyContext)

Required Parameter Group:	
1	Key context token
Input	Char(8)
2	Error code
I/O	Char(*)

Service Program Name: QC3CTX

Default Public Authority: *USE

Threadsafe: Yes

The **Destroy Key Context (OPM, QC3DESKX; ILE, Qc3DestroyKeyContext) API** destroys the key context created with the "Create Key Context (QC3CRTKX, Qc3CreateKeyContext)" on page 78.

Authorities and Locks

Required API authority
*USE

Required Parameter Group

Key context token
INPUT; CHAR(8)

The token of the key context to destroy.

Error code
I/O; CHAR(*)

The structure in which to return error information.
For the format of the structure, see Error Code Parameter.

Error Messages

Message ID	Error Message Text
CPF3C1E E	Required parameter &1 omitted.
CPF3CF1 E	Error code parameter not valid.
CPF3CF2 E	Error(s) occurred during running of &1 API.
CPF9872 E	Program or service program &1 in library &2 ended. Reason code &3.
CPF9DDA E	Unexpected return code &1.
CPF9DF4 E	The key context token does not reference a valid key context.
CPF9DF5 E	The key context is not found or was previously destroyed.

API introduced: V5R3

Top | "Cryptographic Services APIs," on page 1 | APIs by category

Concepts

These are the concepts for this category.

OS/400 and 2058 Cryptographic Function Comparison

The following table lists what cryptographic functions are available in OS/400(R) and on the 2058 through the Cryptographic Services APIs.

Function	OS/400	2058
Qc3EncryptData, Qc3DecryptData, Qc3TranslateData		
DES ECB	Yes	Yes
DES CBC	Yes	Yes
DES OFB	Yes	No
DES CFB 1-bit	Yes	No
DES CFB 8-bit	Yes	No
DES CFB 64-bit	Yes	No
TDES ECB	Yes	Yes
TDES CBC	Yes	Yes
TDES OFB	Yes	No
TDES CFB 1-bit	Yes	No
TDES CFB 8-bit	Yes	No
TDES CFB 64-bit	Yes	No
AES ECB	Yes	No
AES CBC	Yes	No
RC4	Yes	No
RSA	Yes	Yes[1]
Qc3CalculateMAC		
DES	Yes	No
TDES	Yes	No
AES	Yes	No
Qc3CalculateHash		
MD5	Yes	No
SHA-1	Yes	No
SHA-256	Yes	No
SHA-384	Yes	No
SHA-512	Yes	No
Qc3CalculateHMAC		
MD5	Yes	No
SHA-1	Yes	No
Qc3CalculateSignature, Qc3VerifySignature	Yes	Yes[2]
Qc3GenPRNs	Yes	Yes[3]
Qc3GenSymmetricKey	Yes	Yes
Qc3GenPKAKeyPair	Yes	No
Qc3GenDHParms	Yes	No
Qc3GenDHKeyPair	Yes	No
Qc3CalculateDHSecretKey	Yes	Yes

[1]Block formating is done in OS/400.

[2]Only the encryption is done on the 2058. The block formatting and hash functions are done in OS/400.

[3]The OS/400 PRNG will automatically seed from a crypto card if one is available.

Top | Cryptographic Services APIs | APIs by category

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

See Code disclaimer information for information pertaining to code examples.

This scenario demonstrates the following:

- "Key Management," one possible design solution with pseudocode.
- "File Encryption" on page 86, sample pseudocode and programs.

The Situation

Briana is writing an application that handles customer data and accounts receivable. Because of recent privacy legislation, she needs to store the customer data encrypted.

Key Management

Before writing programs for her customer data application, Briana needs to set up some sort of key management. Her key management should include the following:

1. Management of multiple KEKs (key-encrypting keys).
   - A KEK is a key used to encrypt other keys.
   - A KEK makes it easier to securely store and transmit data-encrypting keys.
   - Multiple KEKs help isolate encrypted data. (For example, Briana could use different KEKs when dealing with customer data, supplier data, and company proprietary information.)
2. Protection of KEKs.
   - Although KEKs solve the problem of securely storing and transmitting data-encrypting keys, the KEKs now have similar problems.
   - Secure storage of KEKs is commonly achieved by storing them encrypted under a ″master key″ or variants of a master key. The master key is the only key that is not encrypted and MUST be stored securely (e.g. in a safe).

Briana decides on the following key management design. (Note: The following description is an example design for key management. Your needs may dictate a different design.)

Briana chooses to use a database file to store KEKs. She is careful to set up authority to the KEK file so that only users with a legitimate need can access the KEKs. (If necessary, Briana can create multiple KEK files to further isolate access to the KEKs.) Each record will contain two fields, an encrypted KEK value and a label which is the database key field (not to be confused with a cryptographic key).

Whenever a KEK is created, Briana's key management code will encrypt it under a master key before storing it in the KEK file. To better secure the master key, Briana decides not to store it on the system. Instead, her application will establish the master key interactively at job start up. This means that someone must be charged with knowledge and management of the master key. This person is responsible not only for entering the master key, but also for periodically changing it. In addition, whenever the master key is changed, Briana's application must re-encrypt all the KEKs.

When the master key is entered, Briana's application will create a key context for it using the Create Key Context API. The Create Key Context API stores the value of the key below the MI in a key context. Once the key context for the master key is created, Briana's code can wipe the master key value from program

storage. The Create Key Context API returns to the application a key context token. Briana's application will pass the master key context token to other programs within the job that need to handle KEKs. The key context, including the key value, is destroyed with the Destroy Key Context API or at job end.

Among the key management programs that Briana will write are programs to create, get, change, and delete a KEK in the KEK file.

Create_KEK

The Create_KEK program takes the following parameters:

Input	the master key context token
Input	the KEK file name
Input	a label for the KEK
Input	a clear KEK value, or special value *GEN
Output	a key context token for the created KEK
Output	an algorithm context token for the created KEK

The Create_KEK program performs the following steps:

1. Open the KEK file. (If the KEK file does not exist, create it.)
2. If input special value *GEN
   - Generate a key encrypted under the master key using the "Generate Symmetric Key (QC3GENSK, Qc3GenSymmetricKey)" on page 65, or Generate PKA Key Pair API.

   Else
   - Encrypt the clear KEK value under the master key using the "Encrypt Data (QC3ENCDT, Qc3EncryptData)" on page 10.
   - Wipe the clear KEK value from program storage.
3. Write the API output to the KEK file using the supplied label as the database key field. (Return an error if a record with that label already exists.)
4. Create a key context for the KEK using the "Create Key Context (QC3CRTKX, Qc3CreateKeyContext)" on page 78.
5. Create an algorithm context for the KEK using the "Create Algorithm Context (QC3CRTAX, Qc3CreateAlgorithmContext)" on page 72.
6. Return the KEK key context and algorithm context tokens.

Get_KEK

The Get_KEK program takes the following parameters:

Input	the master key context token
Input	the KEK file name
Input	the label of the KEK
Output	a key context token for the KEK
Output	an algorithm context token for the created KEK

The Get_KEK program performs the following steps:

1. Open the KEK file. (Return an error if it does not exist.)
2. Read record using the supplied label. (Return an error if not found.)
3. Create a key context for the KEK using the "Create Key Context (QC3CRTKX, Qc3CreateKeyContext)" on page 78.
4. Create an algorithm context for the KEK using the "Create Algorithm Context (QC3CRTAX, Qc3CreateAlgorithmContext)" on page 72.
5. Return the KEK key context and algorithm context tokens.

Change_KEK

The Change_KEK program takes the following parameters:

Input	the master key context token
Input	the KEK file name
Input	the label of the KEK
Input	a clear KEK value, or special value *GEN
Output	a key context token for the created KEK
Output	an algorithm context token for the created KEK

The Change_KEK program performs the following steps:

1. Open the KEK file. (Return an error if it does not exist.)
2. Read record using the supplied label. (Return an error if not found.)
3. Create a key context for the old KEK using the "Create Key Context (QC3CRTKX, Qc3CreateKeyContext)" on page 78.
4. If input special value *GEN
   - Generate a key encrypted under the master key using the "Generate Symmetric Key (QC3GENSK, Qc3GenSymmetricKey)" on page 65, or "Generate PKA Key Pair (QC3GENPK, Qc3GenPKAKeyPair)" on page 61.

   Else
   - Encrypt the clear KEK value under the master key using the "Encrypt Data (QC3ENCDT, Qc3EncryptData)" on page 10.
   - Wipe the clear KEK value from program storage.
5. Create a key context for the new KEK using the "Create Key Context (QC3CRTKX, Qc3CreateKeyContext)" on page 78.
6. Create an algorithm context for the new KEK using the "Create Algorithm Context (QC3CRTAX, Qc3CreateAlgorithmContext)" on page 72.
7. Re-encrypt under the new KEK any keys that are encrypted under the old KEK using the "Translate Data (QC3TRNDT, Qc3TranslateData)" on page 19.
8. Update the KEK record with the new encrypted KEK value.
9. Return the new KEK key context and algorithm context tokens.

Delete_KEK

The Delete_KEK program takes the following parameters:

Input	the KEK file name
Input	the label of the KEK

The Delete_KEK program performs the following steps:

1. Open the KEK file. (Return an error if it does not exist.)
2. Delete record using the supplied label. (Return an error if not found.)

File Encryption

Now that Briana has her key management set up, she can write the applications that will handle customer data. Briana will store customer data encrypted in a database file. Each record will represent a different customer. Each record includes a customer unique number which is used as the database key field, an initialization vector which is used in the encryption/decryption operations, the accounts receivable balance, and the encrypted customer identification information.

The following is Briana's DDS for the customer file, which she names CUSDTA.

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
     A* CUSTOMER FILE
     A*
     A          R CUSDTAREC                 TEXT('Customer record')
     A            CUSNUM         8  0       TEXT('Customer number')
     A            IV            16         TEXT('Initialization vector')
     A                                       CCSID(65535)
     A            ARBAL         10  2       TEXT('Accounts receivable balance')
     A            CUSDTA        80         TEXT('Encrypted customer data')
     A                                       CCSID(65535)
     A*                                            20    Name
     A*                                            20    Address
     A*                                            20    City
     A*                                             2    State
     A*                                             5    Zip Code
     A*                                            10    Phone number
     A*                                             3    Pad
     A          K CUSNUM
     A*
```

In addition, Briana's application must keep track of information used to process the file, specifically the key used to encrypt the customer data and the last customer number. Briana decides to keep this information in a separate file called CUSPI. CUSPI is created when CUSDTA is created. Although Briana will store the file key encrypted, she is still careful to restrict authority to CUSPI.

Following is Briana's DDS for the customer processing information file.

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
     A* CUSTOMER PROCESSING INFORMATION
     A*
     A          R CUSPIREC                  TEXT('Customer processing info')
     A            KEY           16         TEXT('Encryption key')
     A                                       CCSID(65535)
     A            LASTCUS        8  0       TEXT('Last customer number')
     A*
```

Brianna will need to perform some setup and initialization. Her setup program will

1. Create the CUSDTA and CUSPI files.
2. Call Create_KEK to create a new KEK with label CUSDTA.
3. Call "Generate Symmetric Key (QC3GENSK, Qc3GenSymmetricKey)" on page 65 to generate a file key encrypted under the CUSDTA KEK.
4. Write a record (containing the encrypted file key and last customer number set to 0) to CUSPI.

Briana's application includes a program that writes customer data to the CUSDTA file, and a program that bills customers. These programs are described below. Code examples for these programs are also provided. The code examples illustrate use of the cryptographic APIs for writing encrypted data to a file and reading encrypted data from a file.

Write_Cus

The Write_Cus program takes the following parameters:

Input the master key context token

The Write_Cus program performs the following steps:

1. Open the customer processing information file, CUSPI. (Return an error if the file does not exist.)
2. Read the first (and only) record from CUSPI to retrieve the encrypted file key and last customer number. (Return an error if record not found.)
3. Open the customer file, CUSDTA, for update. (Return an error if the file does not exist.)
4. Call Get_KEK to retrieve the CUSDTA KEK key and algorithm context tokens.

5. Create a key context for the file key using the "Create Key Context (QC3CRTKX, Qc3CreateKeyContext)" on page 78.
6. Setup the algorithm description.
7. Call Get_Customer_Info to retrieve customer information and customer number. (If customer number = 0, it is a new customer. If customer number = 99999999, end the application.)
8. While customer number != 99999999.
9. Generate an IV using the "Generate Pseudorandom Numbers (QC3GENRN,Qc3GenPRNs) API" on page 70.
10. Encrypt the customer data using the "Encrypt Data (QC3ENCDT, Qc3EncryptData)" on page 10.
    - If customer number = 0
      - Add one to last customer number.
      - Set customer number to last customer number.
      - Write the new record to CUSDTA file.

      Else
      - Read CUSDTA record using customer number as the database key. (Return error if record not found.)
      - Update record.
    - Call Get_Customer_Info.
11. Update last customer number in CUSPI.
12. Destroy key context using the "Destroy Key Context (QC3DESKX, Qc3DestroyKeyContext)" on page 82.
13. Close CUSDTA and CUSPI files.

Example

Here are example programs for Write_Cus.

- "Example in ILE C: Writing encrypted data to a file" on page 89
- "Example in ILE RPG: Writing encrypted data to a file" on page 95

Bill_Cus

The Bill_Cus program takes the following parameters:

Input the master key context token

The Bill_Cus program performs the following steps:

1. Open the customer processing information file, CUSPI. (Return an error if the file does not exist.)
2. Read the first (and only) record from CUSPI to retrieve the encrypted file key. (Return an error if record not found.)
3. Close CUSDTA file.
4. Open the customer file, CUSDTA, for sequential read.
5. Call Get_KEK to retrieve the KEK key and algorithm context tokens.
6. Create a key context for the file key using the "Create Key Context (QC3CRTKX, Qc3CreateKeyContext)" on page 78.
7. Setup the algorithm description.
8. While not EOF
   - Read next record.
   - If accounts receivable balance > 0
     - Decrypt customer data using the "Decrypt Data (QC3DECDT, Qc3DecryptData)" on page 2.
     - Call Create_Bill, passing in the decrypted customer data and balance.

9. Close CUSDTA file.
10. Destroy key context using the “Destroy Key Context (QC3DESKX, Qc3DestroyKeyContext)” on page 82.

Examples

Here are example programs for Bill_Cus.

- “Example in ILE C: Reading encrypted data from a file” on page 99
- “Example in ILE RPG: Reading encrypted data from a file” on page 103

Example in ILE C: Writing encrypted data to a file

See Code disclaimer information for information pertaining to code examples.

Refer to “Scenario: Key Management and File Encryption Using the Cryptographic Services APIs” on page 84 for a description of this scenario.

```
/*------------------------------------------------------------------*/
/*                                                                  */
/* Sample C program:  Write_Cus                                     */
/*                                                                  */
/*  COPYRIGHT      5722-SS1 (c) IBM Corp 2004                       */
/*                                                                  */
/*  This material contains programming source code for your         */
/*  consideration.  These examples have not been thoroughly         */
/*  tested under all conditions.  IBM, therefore, cannot            */
/*  guarantee or imply reliability, serviceability, or function     */
/*  of these programs.  All programs contained herein are           */
/*  provided to you "AS IS". THE IMPLIED WARRANTIES OF              */
/*  MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE        */
/*  EXPRESSLY DISCLAIMED. IBM provides no program services for      */
/*  these programs and files.                                       */
/*                                                                  */
/* Description:                                                     */
/*  This is a sample program to demonstrate use of the Cryptographic */
/*  Services APIs.  APIs demonstrated in this program are:          */
/*   Create Key Context                                             */
/*   Generate Pseudorandom Numbers                                  */
/*   Encrypt Data                                                   */
/*   Destroy Key Context                                            */
/*                                                                  */
/* Function:                                                        */
/*  Get customer information, encrypt it, and write it to the       */
/*  Customer Data file (CUSDTA).  The file key is kept in the       */
/*  Customer Processing Information file (CUSPI).                   */
/*                                                                  */
/*  Refer to the iSeries (TM) Information Center for a full         */
/*  description of this scenario.                                   */
/*                                                                  */
/* Parameters:                                                      */
/*  char * 8-byte master key context token                          */
/*                                                                  */
/* Use the following command to compile this program:              */
/*   CRTCMOD MODULE(MY_LIB/WRITE_CUS) SRCFILE(MY_LIB/MY_SRC)        */
/*                                                                  */
/*------------------------------------------------------------------*/

/*------------------------------------------------------------------*/
```

```
/* Retrieve various structures/utilities.                           */
/*------------------------------------------------------------------*/

#include <stdio.h>                    /* Standard I/O header        */
#include <stdlib.h>                   /* General utilities          */
#include <stddef.h>                   /* Standard definitions       */
#include <string.h>                   /* String handling utilities  */
#include <recio.h>                    /* Record I/O routines        */
#include <qusec.h>                    /* Error code structure       */
#include <qc3ctx.h>                   /* Hdr file for Context APIs  */
#include <qc3prng.h>                  /* Hdr file for PRNG APIs     */
#include <qc3dtaen.h>                 /* Hdr file for Encrypt Dta API*/

/*------------------------------------------------------------------*/
/* The following structures were generated with GENCSRC.            */
/*------------------------------------------------------------------*/

  #ifdef __cplusplus
  #include <bcd.h>
  #else
  #include <decimal.h>
  #endif
  /* ---------------------------------------------------- *
  // PHYSICAL FILE : MY_LIB/CUSPI
  // FILE LAST CHANGE DATE : 2004/02/11
  // RECORD FORMAT : CUSPIREC
  // FORMAT LEVEL IDENTIFIER : 248C15A88E09C
   * ---------------------------------------------------- */
  typedef _Packed struct {
     char KEY[16];                    /* ENCRYPTION KEY */

  #ifndef __cplusplus
     decimal( 8, 0) LASTCUS;
  #else
     _DecimalT< 8, 0> LASTCUS;        /* LAST CUSTOMER NUMBER */
                                      /* BCD class SPECIFIED IN DDS */
  #endif
  } CUSPIREC_both_t;

  /* ---------------------------------------------------- *
  // PHYSICAL FILE : MY_LIB/CUSDTA
  // FILE LAST CHANGE DATE : 2004/02/11
  // RECORD FORMAT : CUSDTAREC
  // FORMAT LEVEL IDENTIFIER : 434C857F6F5B3
   * ---------------------------------------------------- */
  typedef _Packed struct {

  #ifndef __cplusplus
     decimal( 8, 0) CUSNUM;
  #else
     _DecimalT< 8, 0> CUSNUM;         /* CUSTOMER NUMBER */
                                      /* BCD class SPECIFIED IN DDS */
  #endif
     char IV[16];                     /* INITIALIZATION VECTOR */

  #ifndef __cplusplus
     decimal(10, 2) ARBAL;
  #else
     _DecimalT<10, 2> ARBAL;          /* ACCOUNTS RECEIVABLE BALANCE */
                                      /* BCD class SPECIFIED IN DDS */
  #endif
     char ECUSDTA[80];                /* ENCRYPTED CUSTOMER DATA */
  } CUSDTAREC_both_t;

/*------------------------------------------------------------------*/
/* Function declarations                                            */
/*------------------------------------------------------------------*/
```

```
                                    /* Get a Key-Encrypting Key    */
int Get_KEK(char *masterKeyToken, char *fileName, char *kekName,
            char *kekToken, char *keaToken);
                                    /* Get a customer information  */
void Get_Customer_Info(char *customerInfo,
                       decimal(8, 0) *customerNumber);

/*--------------------------------------------------------------------*/
/* Start of mainline code.                                            */
/*--------------------------------------------------------------------*/

int Write_Cus(char *mstkeytkn)
{

/*--------------------------------------------------------------------*/
/* Return codes                                                       */
/*--------------------------------------------------------------------*/

  int                   rtn;          /* Return code                */
  #define               ERROR  -1
  #define               OK      0

/*--------------------------------------------------------------------*/
/* File handling variables                                            */
/*--------------------------------------------------------------------*/

  _RFILE               *cuspiPtr;     /* Pointer to CUSPI  file     */
  _RFILE               *cusdtaPtr;    /* Pointer to CUSDTA file     */
  CUSPIREC_both_t       cuspi;        /* CUSPI record               */
  CUSDTAREC_both_t      cusdtain;     /* CUSDTA input record        */
  CUSDTAREC_both_t      cusdtaout;    /* CUSDTA output record       */

/*--------------------------------------------------------------------*/
/* Parameters needed by the Cryptographic Services APIs               */
/*--------------------------------------------------------------------*/

  int                   keyType;      /* Key type                   */
  int                   keySize;      /* Key size                   */
  char                  keyFormat;    /* Key format                 */
  char                  keyForm;      /* Key form                   */
  Qc3_Format_ALGD0200_T algD;         /* Block cipher alg description*/
  char                  AESkctx[8];   /* AES key context token      */
  char                  kektkn[8];    /* Key-encrypting key ctx token*/
  char                  keatkn[8];    /* Key-encrypting alg ctx token*/
  char                  csp;          /* Crypto service provider    */
  char                  pcusdta[80];  /* Plaintext customer data    */
  int                   cipherLen;    /* Length of ciphertext       */
  int                   plainLen;     /* Length of plaintext        */
  int                   rtnLen;       /* Return length              */
  Qus_EC_t              errCode;      /* Error code structure       */
  char                  PRNtype;      /* PRN type                   */
  char                  PRNparity;    /* PRN parity                 */
  unsigned int          PRNlen;       /* Length of PRN data         */

/*--------------------------------------------------------------------*/
/* Input values from Get Customer Information.                        */
/*--------------------------------------------------------------------*/

  char                  inCusInfo[80];/* Customer Information       */
  decimal(8, 0)         inCusNum;     /* Customer number            */

/*--------------------------------------------------------------------*/
/* Initializations                                                    */
/*--------------------------------------------------------------------*/

                                    /* Init to good return        */
```

```
  rtn = OK;
                                    /* Generate exceptions         */
  memset(&errCode, 0, sizeof(errCode));
                                    /* Use any crypto provider     */
  csp = Qc3_Any_CSP;
                                    /* Set inCusInfo to null       */
  memset(inCusInfo, 0, sizeof(inCusInfo));

/*-------------------------------------------------------------------*/
/* Open Customer Processing Information file.  Read first record     */
/* to obtain the encrypted file key and last customer number.        */
/*-------------------------------------------------------------------*/

                                    /* Open CUSPI file             */
  if ((cuspiPtr = _Ropen("MY_LIB/CUSPI", "rr+, arrseq=Y, riofb=N"))
       == NULL)
  {                                 /* If null ptr returned        */
                                    /* Send error message          */
    printf("Open of Customer Processing Information file (CUSPI) failed.");
    return ERROR;                   /* Return with error           */
  }
                                    /* Read the first(only) record */
                                    /* to get encrypted file key.  */
  if ((_Rreadf(cuspiPtr, &cuspi, sizeof(cuspi), __DFT))->num_bytes
       == EOF)
  {                                 /* If record not found         */
                                    /* Send error message          */
    printf("Customer Processing Information (CUSPI) record missing.");
    _Rclose(cuspiPtr);              /* Close CUSPI file            */
    return ERROR;                   /* Return with error           */
  }

/*-------------------------------------------------------------------*/
/* Open Customer Data file.                                          */
/*-------------------------------------------------------------------*/

                                    /* Open CUSDTA file            */
  if ((cusdtaPtr = _Ropen("MY_LIB/CUSDTA", "rr+, riofb=N"))
       == NULL)
  {                                 /* If null ptr returned        */
                                    /* Send error message          */
    printf("Open of CUSDTA file failed.");
    _Rclose(cuspiPtr);              /* Close CUSPI file            */
    return ERROR;                   /* Return with error           */
  }

/*-------------------------------------------------------------------*/
/* Get the KEK under which the file key is encrypted.                */
/*-------------------------------------------------------------------*/

                                    /* Get KEK for the file key    */
  if((rtn = Get_KEK(mstkeytkn, "MY_LIB/KEK_FILE", "CUSDTA", kektkn, keatkn))
      != 0)
  {                                 /* If Get_KEK fails            */
    _Rclose(cusdtaPtr);             /* Close CUSDTA file           */
    _Rclose(cuspiPtr);              /* Close CUSPI file            */
    return ERROR;                   /* Return with error           */
  }

/*-------------------------------------------------------------------*/
/* Create a key context for the file key.                            */
/*-------------------------------------------------------------------*/

  keySize = sizeof(cuspi.KEY);      /* Key size                    */
  keyFormat = Qc3_Bin_String;       /* Key format is binary string */
  keyType = Qc3_AES;                /* Key type is AES             */
  keyForm = Qc3_Encrypted;          /* Key string is encrypted     */
```

```
                                    /* Create key context          */
  Qc3CreateKeyContext(cuspi.KEY, &keySize, &keyFormat, &keyType,
                      &keyForm, kektkn, keatkn, AESkctx, &errCode);

/*-------------------------------------------------------------------*/
/* Set up algorithm description for encrypting customer data.        */
/*-------------------------------------------------------------------*/

  memset(&algD, 0, sizeof(algD));     /* Init description to null    */
  algD.Block_Cipher_Alg = Qc3_AES;    /* Use AES algorithm           */
  algD.Block_Length = 16;             /* Block size is 16            */
  algD.Mode = Qc3_CBC;                /* Use cipher block chaining   */
  algD.Pad_Option = Qc3_No_Pad;       /* Do not pad                  */

/*-------------------------------------------------------------------*/
/* Get customer information.                                         */
/*-------------------------------------------------------------------*/

                                    /* Get customer information    */
                                    /*  and customer number        */
    Get_Customer_Info(inCusInfo, &inCusNum);

/*-------------------------------------------------------------------*/
/* Repeat loop until no more customers to add/update.                */
/*-------------------------------------------------------------------*/

                                    /* Exit program when customer  */
  while (inCusNum != 99999999)      /*  number = 99999999          */
  {

/*-------------------------------------------------------------------*/
/* Generate an Initialization Vector for the customer.               */
/*-------------------------------------------------------------------*/

    PRNtype = Qc3PRN_TYPE_NORMAL;     /* Generate real random numbers*/
    PRNparity = Qc3PRN_NO_PARITY;     /* Do not adjust parity        */
    PRNlen = 16;                      /* Generate 16 bytes           */
    Qc3GenPRNs(cusdtaout.IV, PRNlen, PRNtype, PRNparity, &errCode);

/*-------------------------------------------------------------------*/
/* Encrypt customer information.                                     */
/*-------------------------------------------------------------------*/

                                    /* Copy IV to alg description  */
    memcpy(algD.Init_Vector, cusdtaout.IV, 16);

                                    /* Encrypt customer data       */
    plainLen = sizeof(inCusInfo);
    cipherLen = sizeof(cusdtaout.ECUSDTA);
    Qc3EncryptData(inCusInfo, &plainLen, Qc3_Data,
                   (unsigned char *)&algD, Qc3_Alg_Block_Cipher,
                   (unsigned char *)&AESkctx, Qc3_Key_Token,
                   &csp, NULL, cusdtaout.ECUSDTA, &cipherLen, &rtnLen, &errCode);

/*-------------------------------------------------------------------*/
/* Write customer data to file CUSDTA.                               */
/*-------------------------------------------------------------------*/

    if (inCusNum == 0)                /* If new customer             */
    {
      cuspi.LASTCUS += 1;             /* Increment last customer num */
      cusdtaout.CUSNUM=cuspi.LASTCUS; /* Give new customer a number  */
      cusdtaout.ARBAL = 10;           /* Set balance to setup fee    */
                                      /* Write record to file        */
      if ((_Rwrite(cusdtaPtr, &cusdtaout, sizeof(cusdtaout)))->num_bytes
              < sizeof(cusdtaout))
      {                               /* If write fails              */
```

```
                                    /* Send error message          */
        printf("Error occurred writing record to CUSDTA file.");
        inCusNum = 99999999;        /* Set to exit loop            */
        rtn = ERROR;                /* Indicate error condition    */
      }
    }
    else                            /* If existing customer        */
    {
                                    /* Read existing record        */
      if ((_Rreadk(cusdtaPtr, &cusdtain, sizeof(cusdtain), __KEY_EQ,
           &inCusNum, sizeof(inCusNum))) -> num_bytes < sizeof(cusdtain))
      {                             /* If read fails               */
                                    /* Send error message          */
        printf("Error occurred reading record in CUSDTA file.");
        inCusNum = 99999999;        /* Set to exit loop            */
        rtn = ERROR;                /* Indicate error condition    */
      }
                                    /* Copy customer number        */
      cusdtaout.CUSNUM = cusdtain.CUSNUM;
                                    /* Copy balance                */
      cusdtaout.ARBAL = cusdtain.ARBAL;
                                    /* Update customer record      */
      if ((_Rupdate(cusdtaPtr, &cusdtaout, sizeof(cusdtaout)))->num_bytes
              < sizeof(cusdtaout))
      {                             /* If update fails             */
                                    /* Send error message          */
        printf("Error occurred updating record in CUSDTA file.");
        inCusNum = 99999999;        /* Set to exit loop            */
        rtn = ERROR;                /* Indicate error condition    */
      }
    }

/*-------------------------------------------------------------------*/
/* Get customer information.                                         */
/*-------------------------------------------------------------------*/

                                    /* Get customer information    */
    if (rtn == OK)                  /*  and customer number        */
      Get_Customer_Info(inCusInfo, &inCusNum);

  }                                 /* Return to top of while loop */


/*-------------------------------------------------------------------*/
/* Update last customer number in CUSPI file.                        */
/*-------------------------------------------------------------------*/

                                    /* Write record to file        */
  if ((_Rwrite(cuspiPtr, &cuspi, sizeof(cuspi)))->num_bytes
         < sizeof(cuspi))
  {                                 /* If write fails              */
                                    /* Send error message          */
    printf("Error occurred updating record in CUSPI file.");
  }

/*-------------------------------------------------------------------*/
/* Cleanup.                                                          */
/*-------------------------------------------------------------------*/

                                    /* Clear plaintext data        */
  memset(inCusInfo, 0, sizeof(inCusInfo));
                                    /* Destroy file key context    */
  Qc3DestroyKeyContext(AESkctx, &errCode);
                                    /* Close CUSDTA file           */
  _Rclose(cusdtaPtr);
                                    /* Close CUSPI file            */
  _Rclose(cuspiPtr);
```

```
                                    /* Return                     */
  return rtn;

}
```

Top | Cryptographic Services APIs | APIs by category

Example in ILE RPG: Writing encrypted data to a file

See Code disclaimer information for information pertaining to code examples.

Refer to “Scenario: Key Management and File Encryption Using the Cryptographic Services APIs” on page 84 for a description of this scenario.

```
     * Sample RPG program: write_cus
     *
     * COPYRIGHT 5722-SS1 (c) IBM Corp 2004
     *
     * This material contains programming source code for your
     * consideration.  These examples have not been thoroughly
     * tested under all conditions.  IBM, therefore, cannot
     * guarantee or imply reliability, serviceability, or function
     * of these programs.  All programs contained herein are
     * provided to you "AS IS".  THE IMPLIED WARRANTIES OF
     * MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
     * EXPRESSLY DISCLAIMED.  IBM provides no program services for
     * these programs and files.
     *
     * Description: This is a sample program to demonstrate use
     * of the Cryptographic Services APIs.  APIs demonstrated in
     * this program are:
     *      Create Key Context
     *      Generate Pseudorandom Numbers
     *      Encrypt Data
     *      Destroy Key Context
     *
     * Function: Get customer information, encrypt it, and write it
     * to the Customer Data file (CUSDTA).  The file key is kept
     * in the Customer Processing Information file (CUSPI).
     *
     * Refer to the iSeries (TM) Information Center for a full
     * description of this scenario.
     *
     * Parameters: 8-byte master key context token
     *
     * Use the following command to compile this program:
     * CRTRPGMOD MODULE(MY_LIB/WRITE_CUS) SRCFILE(MY_LIB/QRPGLESRC)
     *
    H nomain bnddir('QC2LE')

    Fcuspi     uf   e             disk    usropn
    Fcusdta    uf a e             disk    prefix(C) usropn

     * System includes
    D/Copy QSYSINC/QRPGLESRC,QUSEC
    D/Copy QSYSINC/QRPGLESRC,QC3CCI

     * Prototypes
    DWrite_Cus        pr            10i 0 extproc('Write_Cus')
    D keyTkn                         8    const
```

```
DGet_KEK          pr            10i 0 extproc('Get_KEK')
D keyTkn                         1    const
D fileName                       1    const
D kekName                        1    const
D kekKCtx                        8
D kekACtx                        8

D Get_Customer_Info...
D                 pr                  extproc('Get_Customer_Info')
D inCusInfo                      1
D inCusNbr                       8  0

DCrtKeyCtx        pr                  extproc('Qc3CreateKeyContext')
D key                            1    const
D keySize                       10i 0 const
D keyFormat                      1    const
D keyType                       10i 0 const
D keyForm                        1    const
D keyEncKey                      8    const options(*omit)
D keyEncAlg                      8    const options(*omit)
D keyTkn                         8
D errCod                         1

DDestroyKeyCtx    pr                  extproc('Qc3DestroyKeyContext')
D keyTkn                         8    const
D errCod                         1

DEncryptData      pr                  extproc('Qc3EncryptData')
D clrData                        1    const
D clrDataSize                   10i 0 const
D clrDataFmt                     8    const
D algDesc                        1    const
D algDescFmt                     8    const
D keyDesc                        1    const
D keyDescFmt                     8    const
D csp                            1    const
D cspDevNam                     10    const options(*omit)
D EncDta                         1
D DtaLenPrv                     10i 0 const
D DtaLenRtn                     10i 0
D errCod                         1

DGenPRN           pr                  extproc('Qc3GenPRNs')
D prnData                        1
D prnLen                        10i 0 const
D prnType                        1    const
D prnParity                      1    const
D errCod                         1

DPrint            pr            10i 0 extproc('printf')
D charString                      *   value options(*string :*nopass)

PWrite_Cus        b                   export
DWrite_Cus        pi            10i 0
D mstKeyTkn                      8    const

 * Local variable
D csp             s                   1    inz('0')                          Use any crypto prv
D prnType         s                   1    inz('0')                          Real PRNs
D prnParity       s                   1    inz('0')                          No PRN Parity
D error           s             10i 0 inz(-1)                               Error value to rtn
D ok              s             10i 0 inz(0)                                OK value to return
D rtn             s             10i 0                                       Return code
D rtnLen          s             10i 0
D plainLen        s             10i 0
D cipherLen       s             10i 0
D prnLen          s             10i 0 inz(%size(EIV))
```

```
D kekTkn          s              8                                      Key Ctx token
D keaTkn          s              8                                      Alg Ctx Token
D AESkctx         s              8
D keySize         s             10i 0
D keyType         s             10i 0
D keyFormat       s              1
D keyForm         s              1
D inCusInfo       s             80
D inCusNum        s              8  0
D ECUSDTA         s                   like(CCUSDTA)
D EIV             s                   like(CIV)
D NL              c                   x'15'

C                   eval      rtn = ok                                 Init to good return
C                   eval      QUSBPRV = 0                              Use exceptions
 * Open CUSPI file
C                   open(e)   cuspi
C                   if        %error = '1'
C                   callp     Print('Open of Customer Processing -
C                             Information File (CUSPI) failed' + NL)
C                   return    error
C                   endif
 * Read first (only) record to get encrypted file key
C                   read(e)   cuspirec
C                   if        %eof = '1'
C                   callp     Print('Customer Processing Information -
C                             (CUSPI) record missing' + NL)
C                   close     cuspi
C                   return    error
C                   endif
 * Open CUSDTA
C                   open(e)   cusdta
C                   if        %error = '1'
C                   callp     Print('Open of CUSDTA file failed' + NL)
C                   close     cuspi
C                   return    error
C                   endif
 * Get the KEK under which the file key is encrypted
C                   eval      rtn = Get_KEK( mstKeyTkn
C                                            :'MY_LIB/KEK_FILE'
C                                            :'CUSDTA' :kekTkn :keaTkn)
C                   if        rtn <> 0
C                   close     cusdta
C                   close     cuspi
C                   return    error
C                   endif
 * Create a key context for the file key
C                   eval      keySize = %size(KEY)
C                   eval      keyFormat = '0'                          Binary string
C                   eval      keyType = 22                             Key type AES
C                   eval      keyForm = '1'                            Key string encrypted
C                   callp     CrtKeyCtx( KEY     :keySize  :keyFormat
C                                        :keyType :keyForm  :kekTkn
C                                        :keaTkn  :AESkctx  :QUSEC)
 * Set up algorithm description for encrypting customer data
C                   eval      QC3D0200 = *loval                        Init QC3D0200
C                   eval      QC3BCA = keyType                         Block cipher AES
C                   eval      QC3BL = 16                               Block length 16
C                   eval      QC3MODE = '1'                            CBC mode
C                   eval      QC3PO = '0'                              No pad
 * Get customer information and customer number
C                   callp     Get_Customer_Info(inCusInfo :inCusNum)
 * Repeat loop until no more customers to add/update
C                   dow       inCusNum <> 99999999
 * Generate an Initialization Vector for the customer
C                   callp     GenPRN( EIV     :prnLen
C                                     :prnType :prnParity :QUSEC)
```

```
 * Encrypt customer information
C                   eval      QC3IV = EIV
C                   eval      plainLen = %size(inCusInfo)
C                   eval      cipherLen = %size(ECUSDTA)
C                   callp     EncryptData( inCusInfo  :plainLen
C                                          :'DATA0100' :QC3D0200
C                                          :'ALGD0200' :AESkctx
C                                          :'KEYD0100' :csp
C                                          :*OMIT      :ECUSDTA
C                                          :cipherLen  :rtnLen
C                                          :QUSEC)
 * Write customer data to file CUSDTA if new customer
C                   if        inCusNum = 0                                  New customer?
C                   eval      LASTCUS += 1                                  Increment last CusNo
C                   eval      CCUSNUM = LASTCUS                             Give new number
C                   eval      CARBAL = 10                                   Set to setup fee
C                   eval      CIV = EIV
C                   eval      CCUSDTA = ECUSDTA
C                   write(e)  cusdtarec
C                   if        %error = '1'
C                   callp     Print('Error occurred writing -
C                             record to CUSDTA file' + NL)
C                   eval      inCusNum = 99999999                           Set to exit loop
C                   eval      rtn = error                                   Indicate error
C                   endif
C                   else                                                    Existing customer
 * Read existing customer
C     inCusNum      chain(e)  cusdtarec
C                   if        %error = '1'
C                   callp     Print('Error occurred reading -
C                             record in CUSDTA file' + NL)
C                   eval      inCusNum = 99999999                           Set to exit loop
C                   eval      rtn = error                                   Indicate error
C                   endif
C                   eval      CCUSDTA = ECUSDTA
C                   eval      CIV = EIV
C                   update(e) cusdtarec
C                   if        %error = '1'
C                   callp     Print('Error occurred updating -
C                             record in CUSDTA file' + NL)
C                   eval      inCusNum = 99999999                           Set to exit loop
C                   eval      rtn = error                                   Indicate error
C                   endif
C                   endif
C                   if        rtn = ok
C                   callp     Get_Customer_Info(inCusInfo :inCusNum)
C                   endif
C                   enddo
C                   update(e) cuspirec                                      Update last custno
C                   if        %error = '1'
C                   callp     Print('Error occurred updating -
C                             record in CUSPI file' + NL)
C                   endif
 * Cleanup
C                   callp     DestroyKeyCtx( AESkctx :QUSEC)                Destroy file key ctx
C                   close     cusdta
C                   close     cuspi
C                   return    rtn
P                 e
```

Example in ILE C: Reading encrypted data from a file

See Code disclaimer information for information pertaining to code examples.

Refer to "Scenario: Key Management and File Encryption Using the Cryptographic Services APIs" on page 84 for a description of this scenario.

```
/*--------------------------------------------------------------------*/
/*                                                                    */
/* Sample C program:  Bill_Cus                                        */
/*                                                                    */
/*  COPYRIGHT      5722-SS1 (c) IBM Corp 2004                         */
/*                                                                    */
/*  This material contains programming source code for your           */
/*  consideration.  These examples have not been thoroughly           */
/*  tested under all conditions.  IBM, therefore, cannot              */
/*  guarantee or imply reliability, serviceability, or function       */
/*  of these programs.  All programs contained herein are             */
/*  provided to you "AS IS". THE IMPLIED WARRANTIES OF                */
/*  MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE          */
/*  EXPRESSLY DISCLAIMED. IBM provides no program services for        */
/*  these programs and files.                                         */
/*                                                                    */
/* Description:                                                       */
/*  This is a sample program to demonstrate use of the Cryptographic  */
/*  Services APIs.  APIs demonstrated in this program are:            */
/*   Create Key Context                                               */
/*   Decrypt Data                                                     */
/*   Destroy Key Context                                              */
/*                                                                    */
/* Function:                                                          */
/*  For each record in the Customer Data file (CUSDTA), check the     */
/*  accounts receivable balance.  If there is a balance, decrypt the  */
/*  customer's data and call Bill_Cus to create a bill.  The customer */
/*  data is encrypted with a file key kept in the Customer Processing */
/*  Information file (CUSPI).                                         */
/*                                                                    */
/*  Refer to the iSeries (TM) Information Center for a full           */
/*  description of this scenario.                                     */
/*                                                                    */
/* Parameters:                                                        */
/*  char * 8-byte master key context token                            */
/*                                                                    */
/* Use the following command to compile this program:                 */
/*   CRTCMOD MODULE(MY_LIB/BILL_CUS) SRCFILE(MY_LIB/MY_SRC)           */
/*                                                                    */
/*--------------------------------------------------------------------*/

/*--------------------------------------------------------------------*/
/* Retrieve various structures/utilities.                             */
/*--------------------------------------------------------------------*/

#include <stdio.h>                   /* Standard I/O header           */
#include <stdlib.h>                  /* General utilities             */
#include <stddef.h>                  /* Standard definitions          */
#include <string.h>                  /* String handling utilities     */
#include <recio.h>                   /* Record I/O routines           */
#include <qusec.h>                   /* Error code structure          */
#include <qc3ctx.h>                  /* Hdr file for Context APIs     */
#include <qc3dtade.h>                /* Hdr file for Decrypt Dta API*/

/*--------------------------------------------------------------------*/
/* The following structures were generated with GENCSRC.              */
/*--------------------------------------------------------------------*/

  #ifdef __cplusplus
  #include <bcd.h>
```

```
  #else
  #include <decimal.h>
  #endif
  /* ---------------------------------------------------- *
  // PHYSICAL FILE : MY_LIB/CUSPI
  // FILE LAST CHANGE DATE : 2004/02/11
  // RECORD FORMAT : CUSPIREC
  // FORMAT LEVEL IDENTIFIER : 248C15A88E09C
   * ---------------------------------------------------- */
  typedef _Packed struct {
     char KEY[16];                        /* ENCRYPTION KEY */

  #ifndef __cplusplus
     decimal( 8, 0) LASTCUS;
  #else
     _DecimalT< 8, 0> LASTCUS;            /* LAST CUSTOMER NUMBER */
                                          /* BCD class SPECIFIED IN DDS */
  #endif
  } CUSPIREC_both_t;

  /* ---------------------------------------------------- *
  // PHYSICAL FILE : MY_LIB/CUSDTA
  // FILE LAST CHANGE DATE : 2004/02/11
  // RECORD FORMAT : CUSDTAREC
  // FORMAT LEVEL IDENTIFIER : 434C857F6F5B3
   * ---------------------------------------------------- */
  typedef _Packed struct {

  #ifndef __cplusplus
     decimal( 8, 0) CUSNUM;
  #else
     _DecimalT< 8, 0> CUSNUM;             /* CUSTOMER NUMBER */
                                          /* BCD class SPECIFIED IN DDS */
  #endif
     char IV[16];                         /* INITIALIZATION VECTOR */

  #ifndef __cplusplus
     decimal(10, 2) ARBAL;
  #else
     _DecimalT<10, 2> ARBAL;              /* ACCOUNTS RECEIVABLE BALANCE */
                                          /* BCD class SPECIFIED IN DDS */
  #endif
     char ECUSDTA[80];                    /* ENCRYPTED CUSTOMER DATA */
  } CUSDTAREC_both_t;

/*--------------------------------------------------------------------*/
/* Function declarations                                              */
/*--------------------------------------------------------------------*/

                                    /* Get a Key-Encrypting Key    */
int Get_KEK(char *masterKeyToken, char *fileName, char *kekName,
            char *kekToken, char *keaToken);
                                    /* Create a bill               */
void Create_Bill(char *customerData, decimal(10, 2) balance);

/*--------------------------------------------------------------------*/
/* Start of mainline code.                                            */
/*--------------------------------------------------------------------*/

int Bill_Cus(char *mstkeytkn)
{

/*--------------------------------------------------------------------*/
/* Return codes                                                       */
/*--------------------------------------------------------------------*/

  int                   rtn;          /* Return code                 */
```

```
  #define               ERROR  -1
  #define               OK      0

/*-------------------------------------------------------------------*/
/* File handling variables                                           */
/*-------------------------------------------------------------------*/

  _RFILE               *cuspiPtr;     /* Pointer to CUSPI  file      */
  _RFILE               *cusdtaPtr;    /* Pointer to CUSDTA file      */
  _RIOFB_T             *cuspifdbk;    /* CUSPI  I/O Feedback         */
  _RIOFB_T             *cusdtafdbk;   /* CUSDTA I/O Feedback         */
  CUSPIREC_both_t       cuspi;        /* CUSPI record                */
  CUSDTAREC_both_t      cusdta;       /* CUSDTA record               */

/*-------------------------------------------------------------------*/
/* Parameters needed by the Cryptographic Services APIs              */
/*-------------------------------------------------------------------*/

  int                   keyType;      /* Key type                    */
  int                   keySize;      /* Key size                    */
  char                  keyFormat;    /* Key format                  */
  char                  keyForm;      /* Key form                    */
  Qc3_Format_ALGD0200_T algD;         /* Block cipher alg description*/
  char                  AESkctx[8];   /* AES key context token       */
  char                  kektkn[8];    /* Key-encrypting key ctx token*/
  char                  keatkn[8];    /* Key-encrypting alg ctx token*/
  char                  csp;          /* Crypto service provider     */
  char                  pcusdta[80];  /* Plaintext customer data     */
  int                   cipherLen;    /* Length of ciphertext        */
  int                   plainLen;     /* Length of plaintext         */
  int                   rtnLen;       /* Return length               */
  Qus_EC_t              errCode;      /* Error code structure        */

/*-------------------------------------------------------------------*/
/* Initializations                                                   */
/*-------------------------------------------------------------------*/

                                      /* Generate exceptions         */
  memset(&errCode, 0, sizeof(errCode));
                                      /* Use any crypto provider     */
  csp = Qc3_Any_CSP;

/*-------------------------------------------------------------------*/
/* Open Customer Processing Information file.  Read first record     */
/* to obtain the encrypted file key.  Close file.                    */
/*-------------------------------------------------------------------*/

                                      /* Open CUSPI file             */
  if ((cuspiPtr = _Ropen("MY_LIB/CUSPI", "rr, arrseq=Y, riofb=N"))
       == NULL)
  {                                   /* If null ptr returned        */
                                      /* Send error message          */
    printf("Open of Customer Processing Information file (CUSPI) failed.");
    return ERROR;                     /* Return with error           */
  }
                                      /* Read the first(only) record */
                                      /* to get encrypted file key.  */
  if ((_Rreadf(cuspiPtr, &cuspi, sizeof(cuspi), __DFT))->num_bytes
       == EOF)
  {                                   /* If record not found         */
                                      /* Send error message          */
    printf("Customer Processing Information (CUSPI) record missing.");
    _Rclose(cuspiPtr);                /* Close CUSPI file            */
    return ERROR;                     /* Return with error           */
  }

  _Rclose(cuspiPtr);                  /* Close CUSPI file            */
```

```
/*-------------------------------------------------------------------*/
/* Open Customer Data file.                                          */
/*-------------------------------------------------------------------*/

                                    /* Open CUSDTA file              */
  if ((cusdtaPtr = _Ropen("MY_LIB/CUSDTA", "rr, arrseq=Y, riofb=N"))
       == NULL)
  {                                 /* If null ptr returned          */
                                    /* Send error message            */
    printf("Open of CUSDTA file failed.");
    return ERROR;
  }

/*-------------------------------------------------------------------*/
/* Get the KEK under which the file key is encrypted.                */
/*-------------------------------------------------------------------*/

                                    /* Get KEK for the file key      */
  if((rtn = Get_KEK(mstkeytkn, "MY_LIB/KEK_FILE", "CUSDTA", kektkn, keatkn))
      != 0)
  {                                 /* If Get_KEK fails              */
                                    /* Send error message            */
    _Rclose(cusdtaPtr);             /* Close CUSDTA file             */
    return ERROR;                   /* Return with error             */
  }

/*-------------------------------------------------------------------*/
/* Create a key context for the file key.                            */
/*-------------------------------------------------------------------*/

  keySize = sizeof(cuspi.KEY);      /* Key size                      */
  keyFormat = Qc3_Bin_String;       /* Key format is binary string   */
  keyType = Qc3_AES;                /* Key type is AES               */
  keyForm = Qc3_Encrypted;          /* Key string is encrypted       */
                                    /* Create key context            */
  Qc3CreateKeyContext(cuspi.KEY, &keySize, &keyFormat, &keyType,
                      &keyForm, kektkn, keatkn, AESkctx, &errCode);

/*-------------------------------------------------------------------*/
/* Set up algorithm description for decrypting customer data.        */
/*-------------------------------------------------------------------*/

  memset(&algD, 0, sizeof(algD));   /* Init to zero                  */
  algD.Block_Cipher_Alg = Qc3_AES;  /* Use AES algorithm             */
  algD.Block_Length = 16;           /* Block size is 16              */
  algD.Mode = Qc3_CBC;              /* Use cipher block chaining     */
  algD.Pad_Option = Qc3_No_Pad;     /* Do not pad                    */

/*-------------------------------------------------------------------*/
/* Read each record of CUSDTA.                                       */
/*-------------------------------------------------------------------*/

                                    /* Read next record in file      */
                                    /* while not End-Of-File         */
  while ((_Rreadn(cusdtaPtr, &cusdta, sizeof(cusdta), __DFT))->num_bytes
          != EOF)
  {

/*-------------------------------------------------------------------*/
/* If accounts receivable balance > 0, decrypt customer data and     */
/* create a bill for the customer.                                   */
/*-------------------------------------------------------------------*/

    if (cusdta.ARBAL > 0)
    {
                                    /* Copy IV to alg description    */
```

```
      memcpy(algD.Init_Vector, cusdta.IV, 16);

                                     /* Decrypt customer data       */
      cipherLen = sizeof(cusdta.ECUSDTA);
      plainLen = sizeof(pcusdta);
      Qc3DecryptData(cusdta.ECUSDTA, &cipherLen,
                     (unsigned char *)&algD, Qc3_Alg_Block_Cipher,
                     (unsigned char *)&AESkctx, Qc3_Key_Token,
                     &csp, NULL, pcusdta, &plainLen, &rtnLen, &errCode);

                                     /* Create bill                 */
      Create_Bill(pcusdta, cusdta.ARBAL);
    }
  }

/*--------------------------------------------------------------------*/
/* Cleanup.                                                           */
/*--------------------------------------------------------------------*/

                                     /* Clear plaintext data        */
  memset(&pcusdta, 0, sizeof(pcusdta));
                                     /* Destroy file key context    */
  Qc3DestroyKeyContext(AESkctx, &errCode);
                                     /* Close CUSDTA file           */
  _Rclose(cusdtaPtr);
                                     /* Return successful           */
  return OK;

}
```

Top | Cryptographic Services APIs | APIs by category

Example in ILE RPG: Reading encrypted data from a file

See Code disclaimer information for information pertaining to code examples.

Refer to “Scenario: Key Management and File Encryption Using the Cryptographic Services APIs” on page 84 for a description of this scenario.

```
      * Sample RPG program: bill_cus
      *
      * COPYRIGHT 5722-SS1 (c) IBM Corp 2004
      *
      * This material contains programming source code for your
      * consideration.  These examples have not been thoroughly
      * tested under all conditions.  IBM, therefore, cannot
      * guarantee or imply reliability, serviceability, or function
      * of these programs.  All programs contained herein are
      * provided to you "AS IS".  THE IMPLIED WARRANTIES OF
      * MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
      * EXPRESSLY DISCLAIMED.  IBM provides no program services for
      * these programs and files.
      *
      * Description: This is a sample program to demonstrate use
      * of the Cryptographic Services APIs.  APIs demonstrated in
      * this program are:
      *      Create Key Context
      *      Decrypt Data
      *      Destroy Key Context
      *
      * Function: For each record in the Customer Data file (CUSDTA),
      * check the accounts receivable balance.  If there is a balance
```

```
 * decrypt the customers data and call bill_cus to create a bill.
 * The customer data is encrypted with a file key kept in the
 * Customer Processing Information file (CUSPI).
 *
 * Refer to the iSeries (TM) Information Center for a full
 * description of this scenario.
 *
 * Parameters: 8-byte master key context token
 *
 * Use the following command to compile this program:
 * CRTRPGMOD MODULE(MY_LIB/BILL_CUS) SRCFILE(MY_LIB/QRPGLESRC)
 *
H nomain bnddir('QC2LE')

Fcuspi     uf   e             disk    usropn
Fcusdta    uf a e             disk    prefix(C) usropn

 * System includes
D/Copy QSYSINC/QRPGLESRC,QUSEC
D/Copy QSYSINC/QRPGLESRC,QC3CCI

 * Prototypes
DBill_Cus         pr            10i 0 extproc('Bill_Cus')
D keyTkn                         8    const

DCreate_Bill      pr            10i 0 extproc('Create_Bill')
D cusDta                           *   value options(*string)
D balance                        10  2 value

DGet_KEK          pr            10i 0 extproc('Get_KEK')
D keyTkn                         1    const
D fileName                       1    const
D kekName                        1    const
D kekKCtx                        8
D kekACtx                        8

DCrtKeyCtx        pr                  extproc('Qc3CreateKeyContext')
D key                            1    const
D keySize                       10i 0 const
D keyFormat                      1    const
D keyType                       10i 0 const
D keyForm                        1    const
D keyEncKey                      8    const options(*omit)
D keyEncAlg                      8    const options(*omit)
D keyTkn                         8
D errCod                         1

DDestroyKeyCtx    pr                  extproc('Qc3DestroyKeyContext')
D keyTkn                         8    const
D errCod                         1

DDecryptData      pr                  extproc('Qc3DecryptData')
D encData                        1    const
D encDataSize                   10i 0 const
D algDesc                        1    const
D algDescFmt                     8    const
D keyDesc                        1    const
D keyDescFmt                     8    const
D csp                            1    const
D cspDevNam                     10    const options(*omit)
D clrDta                         1
D clrLenPrv                     10i 0 const
D clrLenRtn                     10i 0
D errCod                         1

DPrint            pr            10i 0 extproc('printf')
D charString                       *   value options(*string :*nopass)
```

```
PBill_Cus         b                   export
DBill_Cus         pi            10i 0
D mstKeyTkn                      8    const

 * Local variable
D csp             s              1    inz('0')                            Use any crypto prv
D error           s             10i 0 inz(-1)                             Error value
D ok              s             10i 0 inz(0)
D rtn             s             10i 0                                     Return code
D rtnLen          s             10i 0
D plainLen        s             10i 0
D cipherLen       s             10i 0
D kekTkn          s              8                                        Key Ctx token
D keaTkn          s              8                                        Alg Ctx Token
D AESkctx         s              8
D keySize         s             10i 0
D keyType         s             10i 0
D keyFormat       s              1
D keyForm         s              1
D inCusInfo       s             80
D inCusNum        s              8  0
D ECUSDTA         s                   like(CCUSDTA)
D NL              c                   x'15'

C                   eval      QUSBPRV = 0                                 Use exceptions
 * Open CUSPI file
C                   open(e)   cuspi
C                   if        %error = '1'
C                   callp     Print('Open of Customer Processing -
C                             Information File (CUSPI) failed' + NL)
C                   return    error
C                   endif
 * Read first (only) record to get encrypted file key
C                   read(e)   cuspirec
C                   if        %eof = '1'
C                   callp     Print('Customer Processing Information -
C                             (CUSPI) record missing' + NL)
C                   close     cuspi
C                   return    error
C                   endif
C                   close     cuspi
 * Open CUSDTA
C                   open(e)   cusdta
C                   if        %error = '1'
C                   callp     Print('Open of CUSDTA file failed' + NL)
C                   close     cuspi
C                   return    error
C                   endif
 * Get the KEK under which the file key is encrypted
C                   eval      rtn = Get_KEK( mstKeyTkn
C                                            :'MY_LIB/KEK_FILE'
C                                            :'CUSDTA' :kekTkn :keaTkn)
C                   if        rtn <> 0
C                   close     cusdta
C                   return    error
C                   endif
 * Create a key context for the file key
C                   eval      keySize = %size(KEY)
C                   eval      keyFormat = '0'                             Binary string
C                   eval      keyType = 22                                Key type AES
C                   eval      keyForm = '1'                               Key string encrypted
C                   callp     CrtKeyCtx( KEY     :keySize  :keyFormat
C                                        :keyType :keyForm  :kekTkn
C                                        :keaTkn  :AESkctx  :QUSEC)
 * Set up algorithm description for encrypting customer data
C                   eval      QC3D0200 = *loval                           Init QC3D0200
```

```
C                   eval      QC3BCA = keyType                            Block cipher AES
C                   eval      QC3BL = 16                                  Block length 16
C                   eval      QC3MODE = '1'                               CBC mode
C                   eval      QC3PO = '0'                                 No pad
 * Read each record of CUSDTA
C                   read(e)   cusdtarec
C                   dow       %eof <> '1'
 * If accounts receivable balance > 0, decrypt customer data and
 * create a bill
C                   if        CARBAL > 0
 * Decrypt customer information
C                   eval      QC3IV = CIV
C                   eval      plainLen = %size(CCUSDTA)
C                   eval      cipherLen = %size(ECUSDTA)
C                   callp     DecryptData( CCUSDTA    :cipherLen
C                                          :QC3D0200   :'ALGD0200'
C                                          :AESkctx    :'KEYD0100'
C                                          :csp        :*OMIT
C                                          :ECUSDTA    :plainLen
C                                          :rtnLen     :QUSEC)
C                   callp     Create_Bill( ECUSDTA :CARBAL)
C                   endif
C                   read(e)   cusdtarec
C                   enddo
 * Cleanup
C                   callp     DestroyKeyCtx( AESkctx :QUSEC)              Destroy file key ctx
C                   close     cusdta
C                   return    ok

P                 e
```

Top | Cryptographic Services APIs | APIs by category

Appendix. Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

```
IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY  10504-1785
U.S.A.
```

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

```
IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan
```

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Software Interoperability Coordinator, Department YBWA
3605 Highway 52 N
Rochester, MN 55901
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, IBM License Agreement for Machine Code, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Trademarks

The following terms are trademarks of International Business Machines Corporation in the United States, other countries, or both:
Advanced 36
Advanced Function Printing
Advanced Peer-to-Peer Networking
AFP
AIX
AS/400
COBOL/400
CUA
DB2
DB2 Universal Database
Distributed Relational Database Architecture
Domino
DPI

DRDA
eServer
GDDM
IBM
Integrated Language Environment
Intelligent Printer Data Stream
IPDS
iSeries
Lotus Notes
MVS
Netfinity
Net.Data
NetView
Notes
OfficeVision
Operating System/2
Operating System/400
OS/2
OS/400
PartnerWorld
PowerPC
PrintManager
Print Services Facility
RISC System/6000
RPG/400
RS/6000
SAA
SecureWay
System/36
System/370
System/38
System/390
VisualAge
WebSphere
xSeries

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, and service names may be trademarks or service marks of others.

Terms and conditions for downloading and printing publications

Permissions for the use of the information you have selected for download are granted subject to the following terms and conditions and your indication of acceptance thereof.

Personal Use: You may reproduce this information for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative works of this information, or any portion thereof, without the express consent of IBM(R).

Commercial Use: You may reproduce, distribute and display this information solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of this information, or reproduce, distribute or display this information or any portion thereof outside your enterprise, without the express consent of IBM.

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the information or any data, software or other intellectual property contained therein.

IBM reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the information is detrimental to its interest or, as determined by IBM, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations. IBM MAKES NO GUARANTEE ABOUT THE CONTENT OF THIS INFORMATION. THE INFORMATION IS PROVIDED ″AS-IS″ AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

By downloading or printing information from this site, you have indicated your agreement with these terms and conditions.

Code disclaimer information

This document contains programming examples.

SUBJECT TO ANY STATUTORY WARRANTIES WHICH CANNOT BE EXCLUDED, IBM(R), ITS PROGRAM DEVELOPERS AND SUPPLIERS MAKE NO WARRANTIES OR CONDITIONS EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OR CONDITIONS OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT, REGARDING THE PROGRAM OR TECHNICAL SUPPORT, IF ANY.

UNDER NO CIRCUMSTANCES IS IBM, ITS PROGRAM DEVELOPERS OR SUPPLIERS LIABLE FOR ANY OF THE FOLLOWING, EVEN IF INFORMED OF THEIR POSSIBILITY:

1. LOSS OF, OR DAMAGE TO, DATA;
2. SPECIAL, INCIDENTAL, OR INDIRECT DAMAGES, OR FOR ANY ECONOMIC CONSEQUENTIAL DAMAGES; OR
3. LOST PROFITS, BUSINESS, REVENUE, GOODWILL, OR ANTICIPATED SAVINGS.

SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF INCIDENTAL OR CONSEQUENTIAL DAMAGES, SO SOME OR ALL OF THE ABOVE LIMITATIONS OR EXCLUSIONS MAY NOT APPLY TO YOU.

Printed in USA