Application System/400

SC41-3342-00

Cryptographic Support/400

Version 3

Application System/400

SC41-3342-00

Cryptographic Support/400

Version 3

Take Note!

Before using this information and the product it supports, be sure to read the general information under “Notices” on page v.

First Edition (September 1994)

This edition applies to the licensed program IBM Cryptographic Support/400, (Program 5763-CR1), and the IBM Operating System/400, (Program 5763-SS1), Version 3 Release 1 Modification 0, and to all subsequent releases and modifications until otherwise indicated in new editions. Make sure you are using the proper edition for the level of the product.

Full function DES products, which includes the IBM Cryptographic Support/400, are market limited to U.S. firms, their subsidiaries, and banking customers. These limitations affect all countries except Canada. Exceptions to this marketing limitation must be submitted by individual license application through the country export control function. No customer commitment will be made until a specific license is issued.

Order publications through your IBM representative or the IBM branch serving your locality. If you live in the United States, Puerto Rico, or Guam, you can order publications through the IBM Software Manufacturing Company at 800+879-2755. Publications are not stocked at the address given below.

A form for reader comments is provided at the back of this publication. If the form has been removed, you can mail your comments to:

Attn Department 245
IBM Corporation
3605 Highway 52 N
Rochester, MN 55901-9986 USA

or you can fax your comments to:

United States and Canada: 800+937-3430
Other countries: (+1)+507+253-5192

If you have access to Internet, you can send your comments electronically to *idclerk@rchvmw2.vnet.ibm.com*; IBMMAIL, to *ibmmail(usib56rz)*.

When you send information to IBM, you grant IBM a non-exclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you or restricting your use of it.

Contents

Notices

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. 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 of the intellectual property rights of IBM may be used instead of the IBM product, program, or service. The evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, are the responsibility of the user.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Commercial Relations, IBM Corporation, 208 Harbor Drive, Stamford, CT 06904 , U.S.A.

This publication could contain technical inaccuracies or typographical errors.

This publication may refer to products that are announced but not currently available in your country. This publication may also refer to products that have not been announced in your country. IBM makes no commitment to make available any unannounced products referred to herein. The final decision to announce any product is based on IBM's business and technical judgment.

Changes or additions to the text are indicated by a vertical line (|) to the left of the change or addition.

This publication 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.

This publication contains small programs that are furnished by IBM as simple examples to provide an illustration. 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.

Programming Interface Information

Cryptographic Support/400 is intended to help the programmer with data security capabilities for the AS/400 system. This publication contains no programming interfaces for customers.

Trademarks and Service Marks

The following terms, denoted by an asterisk (*) in this publication, are trademarks of the IBM Corporation in the United States or other countries or both:

Application System/400	Operating System/400
AS/400	OS/400
IBM	400

About Cryptographic Support/400 (SC41-3342)

This book describes the data security capabilities of the Cryptographic Support/400 licensed program. It explains how to use the support and provides reference information for programmers. This book may refer to products that are announced but not available yet.

This book contains small programs which are furnished by IBM as simple examples to provide an illustration. 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.

For a list of related publications, see the Bibliography.

Who Should Use This Book

This book is intended for programmers who are responsible for data security on the AS/400 system.

Before reading this book, you should be familiar with the IBM AS/400 Operating System/400 (OS/400 program).

To use some functions of Cryptographic Support/400, you need to know an AS/400 programming language and be able to enter and create a program at an AS/400 work station.

For a quick introduction to cryptography and a summary of the Cryptographic Support/400 function, see Chapter 1, “Introduction to Cryptographic Support/400.”

Chapter 1. Introduction to Cryptographic Support/400

The AS/400*, Operating System/400* (OS/400*) security features provide protection of data in the system. However, data sent or stored outside the system's controlled environment is susceptible to access without permission (see Figure 1-1).

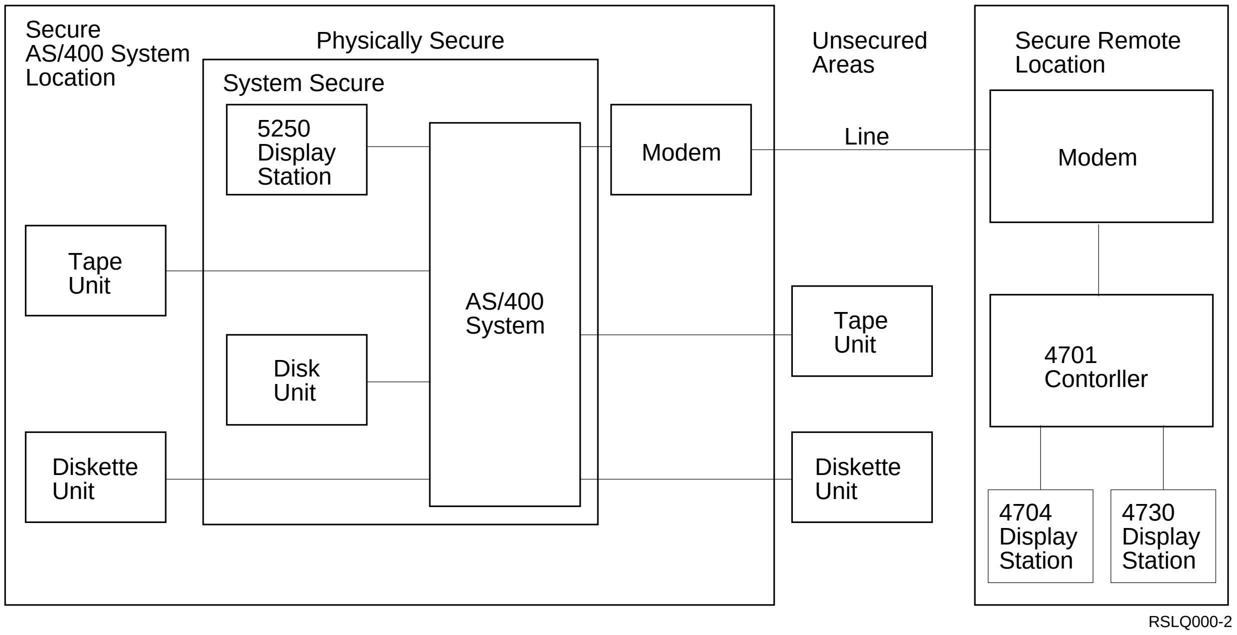


Figure 1-1. Diagram of Data Security Exposures

| Cryptographic Support/400 provides security for data not protected by the OS/400
| licensed program and physical security measures. Cryptographic Support/400 also can add a level of protection to sensitive data stored in the system. This protection can be used for:

- **Communications security** for information traveling across a communications line
- **File security** for data stored on tape, diskette, the system database, or other storage media

The AS/400 system can provide communications security in a financial environment. This book focuses on communications security with the IBM 4700 Finance Communications System. However, both communications and file security procedures are similar. See "File Security Procedures" on page 3-8 for differences and special considerations for file security procedures.

Functions

Cryptographic Support functions fall into three categories:

- **Data encryption and decryption** change plaintext (intelligible data) into ciphertext (unintelligible data) or vice versa through the following:
  - The Data Encryption Algorithm (DEA)
  - Cipher block chaining
  - Data padding

  Data encryption and decryption are also used to generate a message authentication code. You can use the message authentication code to ensure that data was not changed when sent to another location.
- **Key management** protects cryptographic key identities through the use of:
  - Data encrypting key management, including key generation, key encryption, and key translation
  - Cross-domain key management, including maintenance of a cross-domain key table
  - Installing a host system master key
- **Personal identification number (PIN)** functions assign a unique number to a person or organization. There are three PIN functions you can use:
  - Generation
  - Validation
  - Translation

The Cryptographic Process

Cryptographic Support consists of the following parts:

- Licensed internal code, which resides below the machine interface (MI)
- | Cryptographic Support/400 licensed program
- | System/36 environment interface to Cryptographic Support/400

The licensed internal code performs the following functions that must be hidden for security reasons or that require the faster operating speed of the licensed internal code for performance reasons:

- Data encryption using the DEA, cipher block chaining, and padding
- Random key generation
- Key translation
- PIN generation, verification, and translation

| **Note:** A link load of the DEA is performed at install time of the Cryptographic
| Support/400 licensed program. Therefore, if a previous version of licensed internal code is subsequently installed, you will need to re-install the Cryptographic Support licensed program.

The licensed program supplies your link to Cryptographic Support. It consists of:

- A set of control language (CL) commands
- The command processing programs (CPPs)

  Some CL commands may be typed interactively at your work station. Others must be started from a CL program. Some of the CPPs can be called directly by a program written in another AS/400 language.

The System/36 environment interface to Cryptographic Support provides equivalent function for the following System/36 subroutines:

SUBR30	The System/36-compatible RPG II subroutine for cryptography
SUBR31	The System/36-compatible COBOL subroutine for cryptography

Chapter 2. Data Encryption and Decryption

Data encryption is a process of changing plaintext into ciphertext. The decryption process changes data from ciphertext to plaintext.

The Data Encryption Algorithm

The Data Encryption Algorithm (DEA) is the central part of Cryptographic Support. The DEA is a set of mathematical steps that changes plaintext into ciphertext and vice versa. The DEA was adopted from the Data Encryption Standard (DES), used since 1977 for the cryptographic protection of unclassified sensitive data.

Cryptographic Support uses the DEA for three purposes:

- Concealing data
- Authenticating data
- Generating a personal identification number (PIN)

To encrypt data, an application program supplies the DEA with an 8-byte key. The key is presented to the DEA as a 64-bit value. The DEA uses the first 7 bits of each 8-bit byte, for a total of 56 bits, as the key. The remaining 8 bits enforce odd parity of each byte when required. The DEA uses the key and the input data to calculate the output. Given fixed input data, the output is unique for each unique set of 56 bits. Figure 2-1 illustrates this process.

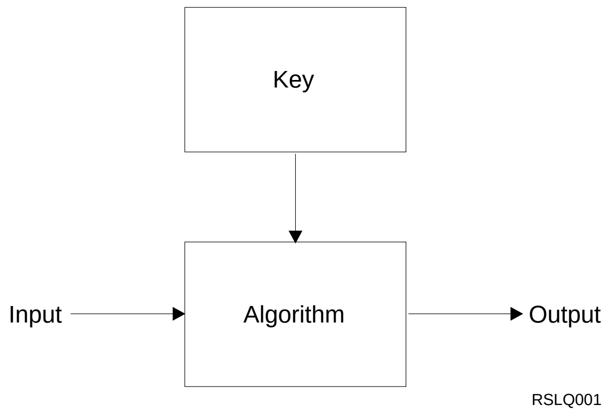


Figure 2-1. DEA Process

Cipher Block Chaining

The DEA encrypts data only in 8-byte blocks. When patterns repeat in the plaintext over multiple 8-byte blocks, these patterns may also occur in the ciphertext, making it more susceptible to analysis. Cipher block chaining is a technique used to obscure and reduce repeated patterns in ciphertext, making it more resistant to analysis.

Cipher block chaining uses a logic operator, exclusive-OR (XOR), to change blocks of data before encryption. The exclusive-OR operation uses an initial chaining value to change the first data block. You provide this chaining value on the encryption request. Cipher block chaining is done by using the exclusive-OR operation on each following 8-byte block of data with the previously encrypted data block.

During the decrypting operation, the process is reversed. Again, the first block of data uses an initial chaining value in the exclusive-OR operation. Decryption with cipher block chaining is done by using the exclusive-OR operation on each block of data with the previous block of encrypted data.

Figure 2-2 and Figure 2-3 illustrate cipher block chaining in the encryption and decryption processes.

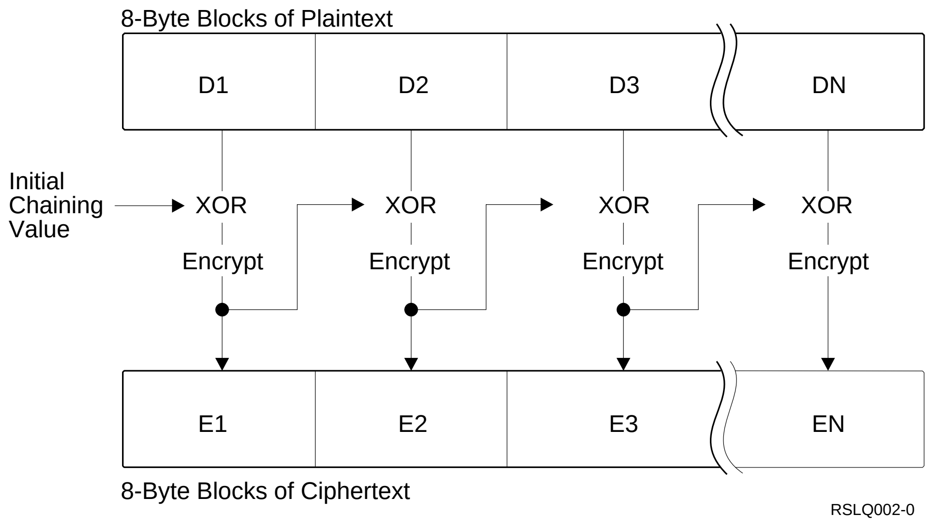


Figure 2-2. Encryption Process

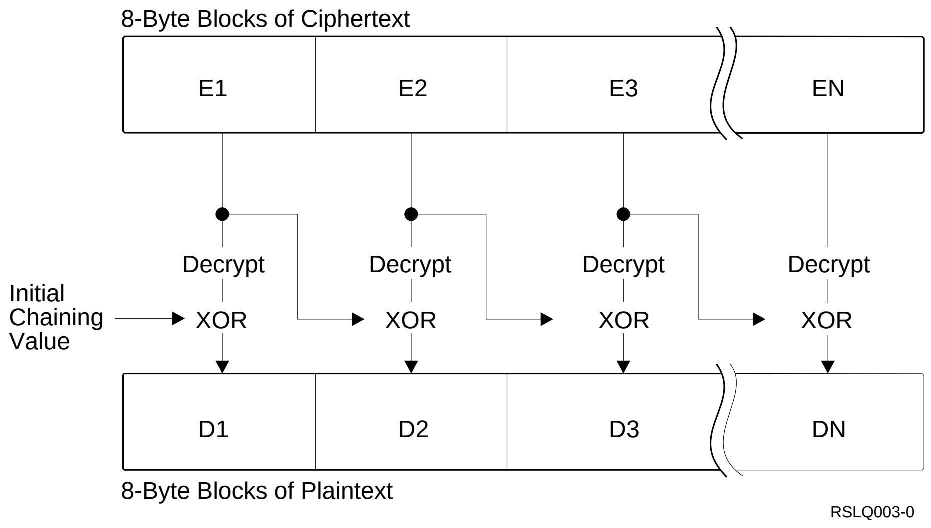


Figure 2-3. Decryption Process

If a block of encrypted data is damaged during sending, only that block and the following block are affected.

When the data length is greater than 8 bytes but not an exact multiple of 8, the last 8-byte encrypted block is encrypted again. The exclusive-OR operation is then used on the first 1-to-7 bytes from this result with the remaining 1-to-7-byte block to produce the last encrypted block. If the data length is fewer than 8 bytes, the initial chaining value is used in place of the previous encrypted block. This process is identical for encrypting or decrypting.

You can indicate whether you want to use cipher block chaining on the encryption or decryption request. If you want to use cipher block chaining on the decryption request, you must know both the key and the initial chaining value used to encrypt the data.

When communicating with a remote application, both the sending and receiving locations must agree on when cipher block chaining will be used. How to keep sending and receiving programs in agreement with a key and an initial chaining value is described in “Communications Security Procedures” on page 3-5.

Padding

The DEA requires an 8-byte block of data to perform encryption. Unless you specify cipher block chaining, the length of the data to be encrypted must be an 8-byte multiple.

Cryptographic Support uses padding to fill a data block containing fewer than 8 bytes with pad characters. When you request padding, supply the pad character, such as a zero or blank, on the encryption request. Cryptographic Support pads to the next 8-byte multiple and records the total number of pad characters used in the last byte of the data block. This count includes the recording byte.

For example, suppose you have 9 bytes of data as shown in Figure 2-4:

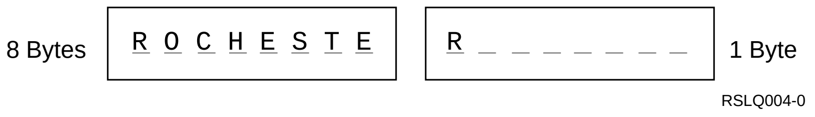


Figure 2-4. Data Block before Padding

If you request padding, the DEA changes the data as shown in Figure 2-5 (where * is the pad character):

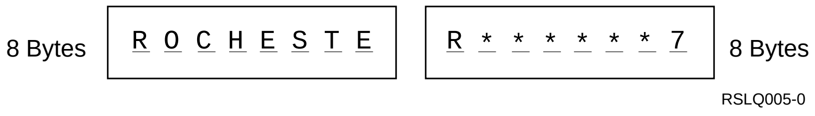


Figure 2-5. Data Block after Padding

Your data decrypting program must indicate that padding has been done on the decryption request. After decryption, Cryptographic Support checks the last byte of data and updates the return data length to reflect the original length of the decrypted data without pad characters. However, the pad characters and pad count remain attached to the data.

Concealing Data

You can use the DEA to conceal data through the Cipher Data (CPHDTA) command or its command processing program (CPP), QCRCIPHR, or by calling the System/36 environment subroutines, SUBR30 or SUBR31. Your user application supplies the variable length of data and the data encrypting key. Additionally, CPHDTA or QCRCIPHR allows you to do the following:

- Request chaining with a specified initial chaining value
- Request padding with a specified pad character
- Supply the data encrypting key in either plaintext or ciphertext

To ensure that data is usable once it is decrypted, the application program performing data encryption must agree with the application program performing the decryption. Consider the following:

- Will data be padded?
- Is cipher block chaining used?
- How will the cryptographic key be selected?
- How will the key be distributed if you are communicating with a remote location?

Authenticating Data

Although you can use the CPHDTA command for data authentication, Cryptographic Support provides the Generate Message Authentication Code (GENMAC) command for verifying data authenticity.

The application program issues the GENMAC command, or calls its CPP, QCRGENMA, directly. QCRGENMA uses cipher block chaining and returns only the last 8 bytes of encrypted data. Generally, only the first 4 bytes are used as the message authentication code (MAC), but all 8 bytes can be used.

You should append the MAC to the end of the encrypted data before sending to another location. The receiving location should generate a separate MAC from the data and compare it with the one received to check for any alterations that may have occurred during sending.

Multiple physical records do not need to be sent individually with a separate MAC for each record. The GENMAC command is issued against each record and uses the encrypted 8 bytes returned as the initial chaining value for the next record. The MAC from the last record is appended to the end of the data as shown in Figure 2-6 on page 2-5.

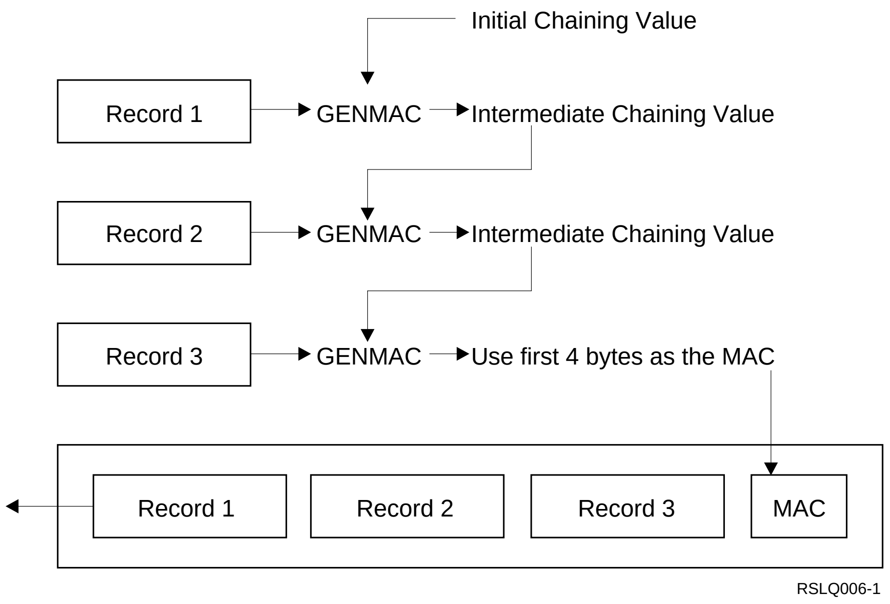


Figure 2-6. Using GENMAC to Send Multiple Records

When encrypting and authenticating a message, the ciphertext must be created before generating the MAC. For security reasons, the key used to encrypt the data and generate the MAC should not be the same.

Chapter 3. Cryptographic Key Management

The degree of protection provided by data encryption depends on the security of the cryptographic keys. Good key management protects cryptographic key identities.

Key Types

Cryptographic Support defines the following two key categories:

- **Key encrypting keys** encrypt other cryptographic keys. Key encrypting keys prevent decryption of the data encrypting key without permission.
- **Data encrypting keys** encrypt data sent across communication lines or stored within a file.

Key Encrypting Keys

Key encrypting keys are used by the system to encrypt other cryptographic keys. The key encrypting keys include the **cross-domain keys** and the **host master key**.

Cross-Domain Keys

Cross-domain keys encrypt data encrypting keys sent to another location or stored in a file. Cross-domain keys allow the frequent exchange of data encrypting keys between two locations. Additionally, using a different cross-domain key at each location protects data at one location from being known at another location without permission.

Each cross-domain key has an associated key name and key use. This allows your program to refer to the key without exposing its actual value. Defining the cross-domain key use also limits the cryptographic functions of that particular key.

There are three types of cross-domain keys:

- **Sending cross-domain keys** encrypt data encrypting keys sent to another location.
- **Receiving cross-domain keys** decrypt data encrypting keys received from another location.
- **Personal identification number (PIN) keys** decrypt PINs and encrypt or decrypt PIN validation data.

| Cryptographic Support/400 provides a file that serves as a table for storing cross-domain keys. This file is called QACRKTBL and is located in the library QUSRSYS. You should save this file to tape or diskette and store it in a secure place. You can use this saved file to restore the file if it is damaged.

The following commands maintain the cross-domain key table:

Command	Function
Add Cross-Domain Key (ADDCRSDMNK)	Used to type in the name, use, and value of a cross-domain key and to add that information to the key table
Change Cross-Domain Key (CHGCRSDMNK)	Changes the value of keys in the table
Remove Cross-Domain Key (RMVCRSDMNK)	Removes any or all keys from the table
Generate Cross-Domain Key (GENCRSDMNK)	Generates random key values and adds those keys to the table

When adding or changing keys in the cross-domain key table, the value you supply must have odd parity in each byte. The resulting sum of all binary digits containing binary 1B must be an odd number.

The parity of the cross-domain key is checked for damage each time it is used. Damaged keys should be removed and added again to the cross-domain key table or restored using the saved file QUSRSYS/QACRKTBL.

Cryptographic Support encrypts the cross-domain keys stored in the table. This prevents anyone who is able to read the table values from using a cross-domain key to decrypt a data encrypting key without permission.

When the Cryptographic Support licensed program is installed on your system for the first time, the cross-domain key table, QACRKTBL, is created in library QUSRSYS. This is done using a prototype file, QACRPTBL, which is copied to QUSRSYS and renamed to QACRKTBL.

If, when installing the Cryptographic Support licensed program, the QACRKTBL already exists in QUSRSYS, the copy does not occur.

If, when installing the Cryptographic Support licensed program, QACRKTBL already exists in QCRP (because your system has an old version of the Cryptographic Support licensed program), QACRKTBL is moved to QUSRSYS.

Note: If any user programs refer to QCRP/QACRKTBL, change them to refer to QUSRSYS/QACRKTBL.

Host Master Key

The host master key is the only cryptographic key not encrypted by the system. You must physically protect the host master key.

Master key **variants** are cryptographic keys used to encrypt the cross-domain keys and are created by altering the master key with an algorithm. This allows the system to act as though several master keys are in use.

There are three variants of the host master key. Each variant encrypts a different cross-domain key type. The original form of the master key is also used to encrypt data encrypting keys. The following table summarizes the use of the master key and its variants:

Master Key Variant	Used to Encrypt
Master Key	Session keys Message authentication keys
Variant 1	Sending cross-domain keys Output PIN protection keys
Variant 2	Receiving cross-domain keys
Variant 3	Input PIN protection keys PIN validation keys

The host master key is stored securely below the AS/400 machine interface. It is accessible only through the supplied Cryptographic Support commands and some service personnel (QSRV and QSRVBAS) tools that dump or display storage. See Chapter 7, "Data Security Considerations," for more information.

For additional security, the system must be in a restricted environment when a new master key is installed. End all subsystems, except the controlling subsystem and one active work station, with the End System (ENDSYS) command.

Cryptographic Support provides the following commands for managing the host master key:

- Set Master Key (SETMSTK) command
- Verify Master Key (VFYMSTK) command
- Change Master Key (CHGMSTK) command

The SETMSTK command sets or changes the value of the host master key. Use this command when first installing the host master key.

The SETMSTK command asks for the values of two 8-byte key parts. By using the exclusive-OR operation on these two values, a host master key is produced. The host master key must have odd parity in every byte. The 8-byte key parts you provide must ensure this.

Separate displays ask for each 8-byte key part. This allows the master key to be installed by two individuals, each having one key part but neither knowing the value of the new master key.

To avoid typing mistakes, each display also asks for an 8-byte binary value key part complement. An exclusive-OR operation on the 8-byte key part and the 8-byte binary complement must produce 8 bytes of hex FF.

The SETMSTK command returns a 2-byte verification code after the new master key has been installed. Record and save this value as it is used with the VFYMSTK and CHGMSTK commands. The VFYMSTK command uses the master key verification code to determine if the host master key has been changed since the verification code was last created.

Like the SETMSTK command, the CHGMSTK command allows you to change the value of the master key. However, the CHGMSTK command requires that you type in the old master key verification code. This permits the security officer (QSECOFR) to allow other user profiles to change the host master key, but only if they know the verification code. You can also use the SETMSTK command to reset the value of the master key if the verification code is lost.

When a SETMSTK or CHGMSTK command is processed, keys stored in the cross-domain key table are automatically encrypted again under the new master key variants. Save the cross-domain key table again after installing a new master key.

When a host master key exists, you can write a control language (CL) program to generate a new master key and its key parts. Use the Generate Cipher Key (GENCPHK) command in this program to generate a random 8-byte value. This is the new master key. Alter the last bit of each byte where necessary to ensure that every byte has odd parity. Use the GENCPHK command again to generate the first key part. To get the second key part, use the exclusive-OR operation on the first key part with the new master key value. If you do not want to use two key parts, specify the master key value for the first key part and 8 bytes of hex 00 for the second key part.

When you change the host master key, save its value in a secure place. You will need this value whenever a new base licensed internal code (VMC) is installed, or if the host master key is damaged. See “Key Security” on page 3-9 for information on restoring the value of the host master key.

Data Encrypting Keys

Use data encrypting keys to encrypt or decrypt sensitive data. When exchanging encrypted data between two locations, the data encrypting key is called a **session key**. See “Exchanging Session Keys” on page 3-5 and “Validating Session Keys” on page 3-6 for further information on session keys.

If you are storing encrypted data, the data encrypting key used to encrypt the data is called a **file key**. See “File Security Procedures” on page 3-8 for more information on file keys.

You can also use a data encrypting key to generate a Message Authentication Code (MAC). “Authenticating Data” on page 2-4 describes how a MAC is used to detect any changes to data during transmission. The data encrypting key used for this purpose is called a **message authentication key.**

The host master key can be used to encrypt data encrypting keys supplied to the Cryptographic Support encryption routines. Encrypted data encrypting keys reduce the possibility of observing the keys through the use of the OS/400 service and debug commands.

Cryptographic Support provides the following two commands to encrypt a data encrypting key under the master key:

- Encrypt Cipher Key (ENCCPHK) command
- Generate Cipher Key (GENCPHK) command

You supply a value to the ENCCPHK command for the data encrypting key in the plaintext. It then returns the value encrypted under the host master key. When you use this command, be sure to protect the plaintext key value from observation without permission.

The GENCPHK command is the preferred method of generating data encrypting keys. This command produces a random key and returns its value encrypted under the host master key. By using the GENCPHK command, your program never needs to handle data encrypting keys in plaintext.

An optional function of the GENCPHK command is to encrypt the value of the random data encrypting key under one or two cross-domain keys. For information on this function, see “Exchanging Session Keys” on page 3-5.

Weak Key Values

When you supply a host master key or cross-domain key, certain values are rejected by Cryptographic Support. These are called **weak** key values because there are known techniques that can be used to decrypt data encrypting keys encrypted under these values.

The following values are considered weak values for the host master key and cross-domain keys:

```
Hex 0101010101010101
Hex FEFEFEFEFEFEFEFE
Hex 1F1F1F1F1E1E1E1E
Hex E0E0E0E0F1F1F1F1
```

The following are also considered weak values for the host master key because of the resulting values of the host master key variants:

```
Hex 8989898989898989
Hex 2323232323232323
Hex 7676767676767676
Hex DCDCDCDCDCDCDCDC
Hex 9797979797979797
Hex 3D3D3D3D2C2C2C2C
Hex 6868686879797979
Hex C2C2C2C2D3D3D3D3
Hex 4545454545454545
Hex BABABABABABABABA
Hex 5B5B5B5B4A4A4A4A
Hex 4A4A4A4B5B5B5B5B
```

Communications Security Procedures

During a cryptographic session, your application program should perform two major key functions to ensure data security. These are the **exchange** and **validation** of session keys.

Exchanging Session Keys

To increase security, generate new session and message authentication keys each time your program begins a session with another program. If the session extends longer than one day, stop the session daily to generate a new key before resuming the exchange of ciphertext.

Sending a session key across communication lines is the most economical means of exchange when new session keys are frequently generated. Encrypting the session key under a cross-domain key protects it from observation on a data link. When you receive a session key from another location, it is encrypted under one of your receiving cross-domain keys. The sending and receiving locations should agree on the value of the cross-domain key used for encrypting the session key before sending the key.

To decrypt the received data, supply the CPHDTA command or its CPP with the session key in either its plaintext or encrypted form. The latter method, which uses a session key encrypted under the host master key, is the most secure method of data decryption.

The Encipher to Master Key (ENCTOMSTK) command converts the session key from encryption under a receiving cross-domain key to encryption under the host master key without exposing the session key in its plaintext form.

To send a session key to another location, encrypt the key under a sending cross-domain key. When encrypting data, supply the CPHDTA command or its CPP with the session key encrypted under the master key. Use either of the following methods to obtain a session key encrypted under a sending cross-domain key:

- The GENCPHK command can return the key value encrypted under the master key and under a cross-domain key. Use the first returned value to encrypt the data and the second returned value as the session key for sending to a remote location.
- The Encipher from Master Key (ENCFRMMSTK) command uses the value of a key encrypted under the host master key as input data. The value of the same key encrypted under a specified sending cross-domain key is returned.

The GENCPHK or ENCFRMMSTK commands are equally secure methods of sending a session key from one location to another as long as the session key is never used in its plaintext form.

Validating Session Keys

Data exchange between two locations may begin once a session key has been distributed. However, if the session key is damaged (such as during transmission or by using an incorrect cross-domain key), retrieving information from the encrypted data is impossible. A key validation procedure prevents this from occurring.

A special session protocol verifies that the application program at each location has an identical data encrypting key. One such protocol requires the exchange of a test message encrypted by both applications. The initiator of the verification procedure decrypts the message twice to confirm its accuracy. The procedure described in “Starting a Cryptographic Session” details this process. Also see the *4700 Finance Communication System Controller Programming Library, Volume 5: Cryptographic Programming* book for another key validation technique.

Starting a Cryptographic Session

Figure 3-1 on page 3-7 shows the start of a cryptographic session, demonstrating session key generation, distribution, and validation. Both locations are AS/400 systems, and both previously agreed on the value of the cross-domain key.

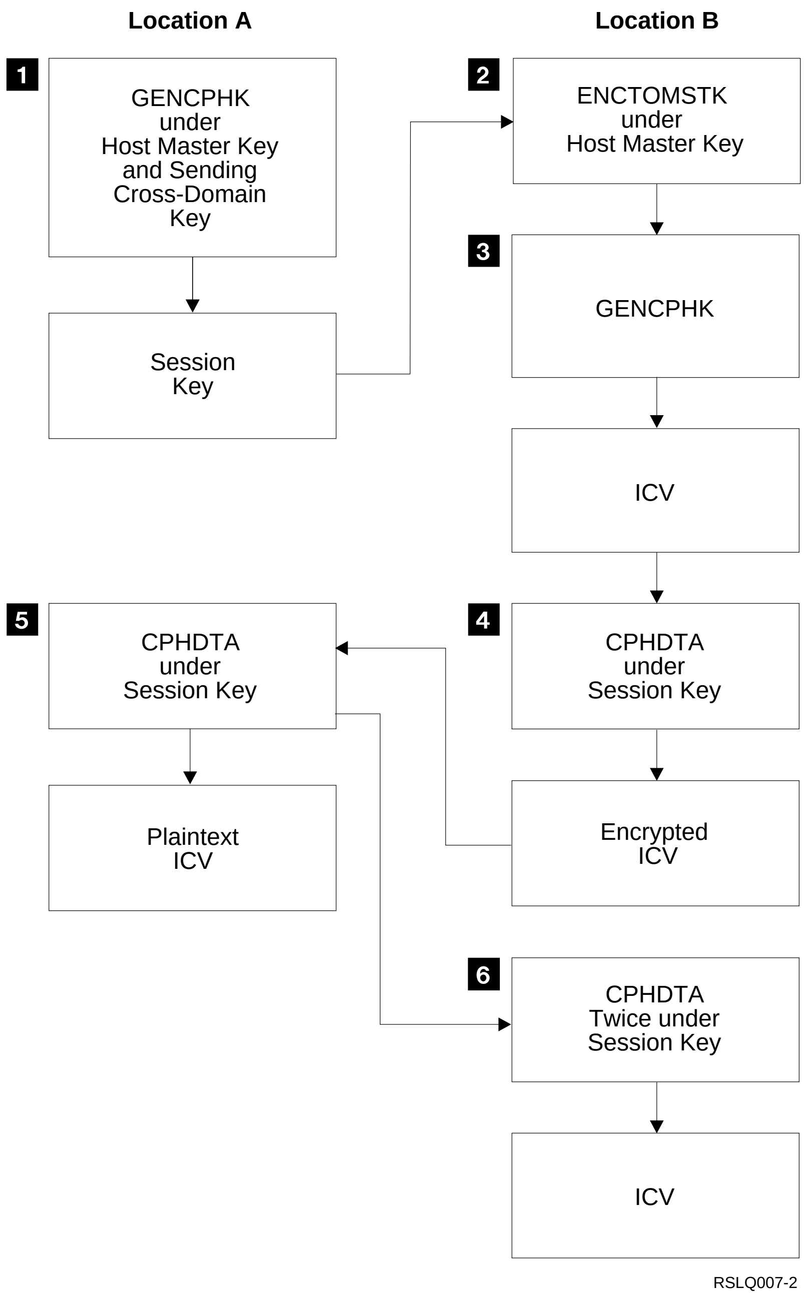


Figure 3-1. Cryptographic Session Flowchart

The following describes the cryptographic session shown in Figure 3-1:

1 Using the GENCPHK command, a random session key is generated and encrypted under the host master key and a sending cross-domain key. The session key encrypted under the cross-domain key is sent to location B.

2 The ENCTOMSTK command receives and decrypts the session key under the receiving cross-domain key and encrypts it again under the host master key.

3 The GENCPHK command generates a random 8-byte value used as a test message and initial chaining value (ICV).

4 The CPHDTA command encrypts the ICV under the session key and sends it to location A.

5 Location A receives the encrypted ICV. It uses the CPHDTA command to again encrypt the ICV under the session key and sends it back to location B.

Location A also decrypts the ICV using the session key to get the plaintext ICV.

6 Location B uses the CPHDTA command to decrypt the ICV twice under the session key. This produces an ICV which is then compared with the original ICV.

File Security Procedures

When storing encrypted data, the file key used to encrypt the data must also be stored so the file can be decrypted later. The file key should be encrypted under a cross-domain key. Access without permission to the data would require knowledge of the secured cross-domain key value.

Do not store the file key encrypted under the host master key, for the following reasons:

- When the host master key value changes, the values of the keys encrypted under the host master key also change. This means the original plaintext value of these keys can no longer be used. The file keys cannot be translated from encryption under the old master key to encryption under the new master key.
- If you recover a file on a different system, you would need to disclose the value of the host master key to that system. However, good key management does not reveal the host master key to another system.
- Knowledge of the file key encrypted under the host master key would enable decryption of the file data by an insider using the CPHDTA command. If the file key is encrypted under a cross-domain key, the person trying to gain access would require the name of the cross-domain key and access to the ENCTOMSTK command or its CPP.

To prevent decryption of the data at your location, encrypt the data encrypting key under a sending cross-domain key using the GENCPHK or ENCFRMMSTK command.

Alternatively, if decryption of the file data is required at your location, encrypt the data encrypting key under a receiving cross-domain key using the GENCPHK command.

Encrypting and Decrypting a File at Two Locations

If system location A wants to encrypt a file to a secondary storage medium for recovery by system location B, both locations must agree on a cross-domain key value.

Location A then does the following:

1. Uses the GENCPHK command to generate a file key encrypted under the host master key and a sending cross-domain key
2. Encrypts the data using the file key encrypted under the host master key
3. Stores the file key encrypted under the sending cross-domain key with the data
4. Destroys the file key encrypted under the host master key

Location B then does the following:

1. Uses the ENCTOMSTK command to translate the file key encrypted under the receiving cross-domain key to encryption under the host master key
2. Decrypts the file data

Encrypting and Decrypting a File at the Same Location

When encrypting and decrypting a file at the same location, use the GENCPHK command to generate a file key encrypted under the receiving cross-domain key and under the host master key.

Use the file key encrypted under the host master key to encrypt the data file. Store the file key encrypted under the receiving cross-domain key with the encrypted data. You can later use that key with the ENCTOMSTK command to get the file key encrypted under the host master key. Then you can decrypt the data.

When you store a file key encrypted under a receiving cross-domain key, someone could obtain the encrypted file data without permission and, with access to the system, use the ENCTOMSTK command to translate the file key and decrypt the file data. Therefore, permission to use the ENCTOMSTK command and its CPP must be tightly controlled.

Key Security

You are responsible for specific security procedures that ensure the security of encrypted data.

Defining a Unique Host Master Key

Your data security depends on the use of a unique host master key. If the same host master key value is used by two locations, the security of the encrypted data at both locations is at risk from data decryption without permission. Therefore, do not use a host master key value that is in use at another location.

Key Selection

Proper key selection limits the success of searches for keys without permission. Consider the following when selecting a key:

- A cryptanalyst may correctly guess the key if a meaningful key (such as the name of a programmer) is used.
- Do not use a known word or name. A computer can try all words in a language dictionary in a matter of minutes.
- Ensure that pseudo-random number generators do not exhibit a predictable pattern.

Use the GENCRSDMNK and GENCPHK commands, instead of organized and predictable methods, to generate all cryptographic keys, including the host master key.

Changing Cryptographic Key Values

Change key values frequently to provide maximum security. Generate a new session key for each session or at least once a day. Change key encrypting keys every 6 to 12 months. Cost, security, and convenience considerations may influence the rate of change.

To change the value of a cross-domain key under which a file key is encrypted, that file must first be decrypted and then encrypted again under a new data encrypting key after the cross-domain key has been changed. No function is provided to decrypt the file's current data encrypting key and encrypt it again under the new cross-domain key. If the file contains a significant amount of encrypted data, this could become a time-consuming procedure. To avoid this problem, you may wish to change cross-domain keys associated with file keys less frequently than you change other cross-domain keys.

Use a given cross-domain key to protect only a few files. This prevents the security of many files from being dependent on the security of a single cross-domain key.

Sending Cryptographic Keys to Other Locations

Cross-domain keys with the same value must be used for cryptographic communication between two locations. One location defines the plain value of the key and sends that value to the other location. Use a secure method of sending the cryptographic key, such as through a bonded courier or registered mail.

Protecting Data Encrypting Keys Encrypted under the Host Master Key

Data encrypting keys encrypted under the host master key are used with the CPHDTA and GENMAC commands to encrypt or decrypt data. The application programs that handle these keys must also protect them with the same level of security provided for plaintext keys.

A data encrypting key encrypted under the host master key is destroyed each time the host master key is changed. To safeguard the data encrypting key, store it encrypted under a cross-domain key that does not change.

Protecting the Host Master Key and the Cross-Domain Key Table

A cross-domain key may be damaged if it is accidentally altered or removed, or if the host master key becomes damaged. If damage occurs, it is impossible to recover the value of the data encrypting keys encrypted under that cross-domain key unless a copy of the cross-domain key exists.

To ensure the integrity of the cross-domain key values, save the cross-domain key table each time you perform the following functions:

- After adding or changing a cross-domain key.
- After installing a new host master key. Since the values in the cross-domain key table are encrypted again under the new variants of the host master key, the new values must be recorded.
- Before restoring library QUSRSYS with an older version of the key table. QACRKTBL is kept in library QUSRSYS, and will be overlaid if QUSRSYS is restored.

To make a copy of the cross-domain key table, save the file QUSRSYS/QACRKTBL to tape or diskette using the SAVOBJ command. Restore the key table using the RSTOBJ command.

Record the value of the host master key and keep it in a secure place for the following reasons:

- If the master key is damaged, the values in the cross-domain key table will not be encrypted again when you install a new host master key. This will corrupt your cross-domain key values.
- The value of the host master key will be overlaid when installing the base licensed internal code. This will also corrupt the cross-domain key values.

In either of these situations, the host master key must be set to its original value before you can read the cross-domain key table. Restoring this value when the host master key is damaged or overlaid will prevent having to manually reinstall all the cross-domain keys. If the saved copy of the cross-domain key table needs to be restored in addition to the master key, use the following procedure:

1. Install the master key.
2. Restore the cross-domain key table.
3. Change the value of the host master key, if desired.

The values in the cross-domain key table will automatically be encrypted again under the new variants of the new host master key.

The cross-domain key table also needs protection from access without permission. See Chapter 7, “Data Security Considerations,” for a discussion of key table authorization.

Other Considerations

When planning key management, consider the following:

- Do not make the keys available in plaintext to any person or process not authorized to use them.
- Record key assignments made to resources or people.
- Encrypt keys kept on the system or sent across data links.

Chapter 4. Personal Identification Number (PIN) Functions

A personal identification number, or PIN, is a unique number assigned to an individual by an organization. PINs are commonly assigned to customers by financial institutions. The PIN is typed at a keyboard and compared with other customer-associated data to provide proof of identity.

PIN Generation Algorithm

A PIN is a unique value produced using an algorithm to alter byte values supplied by the PIN validation key and validation data.

The following steps describe how you use the PIN generation algorithm:

1. Use the Generate Cipher Key (GENCPHK) command to produce the PIN validation key. You must ensure that each byte has odd parity.
2. Get the validation data. This is generally the first 8 bytes from the second track of the customer identification card. If the validation data is fewer than 8 bytes, pad it with a predetermined pad character (for example, blanks). Present these 8 bytes to your program in the form of 16 hexadecimal characters, or have your program change them as shown in Figure 4-1.

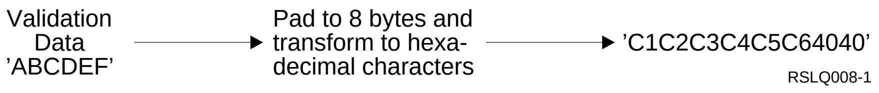


Figure 4-1. Changing Validation Data

3. Generate the PIN value using the GENPIN command or by performing the following steps:

   a. Encrypt the validation data using the PIN validation key and specific encryption on the CPHDTA command, or when calling QCRCIPHR, SUBR30, or SUBR31. The result is 16 hexadecimal characters, as shown in Figure 4-2.

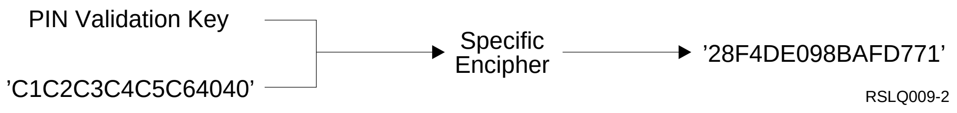


Figure 4-2. Validation Data Encryption

   b. Use a decimalization table to replace each hexadecimal character with a decimal character. The decimalization table consists of 16 decimal characters. Each of the 16 hexadecimal characters that result from step 3 is mapped through the decimalization table and replaced by a decimal character.

In Figure 4-3, the third hexadecimal character, F, indicates that its offset, decimal 15, determines which decimal character occupies the third position of the decimalized string. In this decimalization table, the character located at offset 15 is decimal character 5. Therefore, the third character of the decimal string on the right is 5.

Note: The decimalization table shown here is only an example and should not be misconstrued as the table you should use.

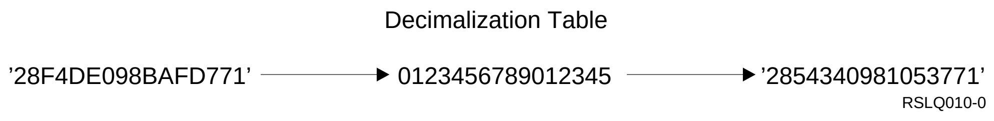


Figure 4-3. Decimalization Table Replacement

4. Use the farthest left x digits, resulting from step 3b, as the intermediate PIN (where x is the defined length of the PIN) as shown below. If the PINs are assigned by a financial institution, use the intermediate PIN as the customer PIN. If the customer has previously selected a PIN value, proceed to step 5.

   x (defined length of PIN) = 4, so intermediate PIN = 2854

5. Calculate the offset data by subtracting the intermediate PIN from the customer-selected PIN value using decimal subtraction with no carry, as illustrated below:

   3000 (customer-selected PIN) – 2854 (intermediate PIN) = 1256 (offset data)

6. Record the offset data where it can be secured by the PIN validation program (generally on the customer's identification card). If only part of the assigned PIN is validated, record the number of PIN digits you are going to validate. Or record the farthest right y digits of the offset data, where y is the PIN check length.

Using the PIN Commands

Cryptographic Support provides three commands for performing PIN operations:

- Generate PIN (GENPIN) command
- Verify PIN (VFYPIN) command
- Translate PIN (TRNPIN) command

The GENPIN command generates a PIN algorithmically related to a given set of validation data. The GENPIN command performs steps 3a and 3b from the algorithm described in "PIN Generation Algorithm" on page 4-1. You must supply the command with the following:

- 16 hexadecimal characters representing 8 bytes of validation data
- The name of a PIN validation key stored in the cross-domain key table
- A decimalization table

The GENPIN command returns 16 decimal digits, of which all or part may be used as the PIN, as illustrated in step 4 in "PIN Generation Algorithm." When using preselected PIN values, your program must calculate and store the offset data as described in steps 5 and 6 in "PIN Generation Algorithm."

The VFYPIN command checks that the correct PIN is used for a given set of validation data. The command generates a PIN from given validation data and compares it against a given PIN value. The result of this comparison is returned in a 1-byte status parameter.

To generate the PIN value from the validation data, the VFYPIN command performs steps 3 through 5 from the algorithm described in “PIN Generation Algorithm” on page 4-1.

In addition to the validation data, an encrypted PIN value, and a status parameter, you must supply the following:

- The protection key under which the PIN is encrypted
- The name of the PIN validation key
- A decimalization table
- A PIN check length

You can also supply the following data to the VFYPIN command:

- A PIN pad character
- The offset data added to the generated PIN before comparison with the given PIN

The encrypted PIN must be supplied as 16-byte hexadecimal characters representing the 8-byte encrypted PIN value. VFYPIN assumes the PIN was in 3624 format before encryption. The 3624 PIN format stores the PIN in 8 bytes with each byte containing 2 digits. If the PIN is not 16 digits long, the remaining digits are padded with a 4-bit value.

For example, assume the PIN value is 5162923 and the padding hexadecimal character is E. Figure 4-4 shows how the PIN looks in 3624 format.

```
8 Bytes  | 51 62 92 3E EE EE EE EE |
                          RSLQ011-0
```

Figure 4-4. Hexadecimal Padding Example

To use this 8-byte value with the VFYPIN command, it must be encrypted under a PIN protection key stored in the cross-domain key table. The encrypted value is supplied to VFYPIN as 16 hexadecimal characters.

Use the CPHDTA command to encrypt the plaintext PIN. When the PIN is in the 8-byte 3624 format, you must change it to 16 hexadecimal characters after encryption. When the plaintext PIN is in the 16-byte 3624 format, select the specific encryption option on the CPHDTA command, which then calculates the changes.

The TRNPIN command decrypts a PIN from encryption under an input PIN protection key to encryption under an output PIN protection key.

Like data encrypting keys, PINs are handled in an encrypted form as much as possible. PINs are encrypted by two types of protection keys:

- **Input PIN protection keys** protect PINs received from another system and PINs handled within the system. When you specify a PIN value for a PIN command parameter, the value must be encrypted under an input PIN protection key.
- **Output PIN protection keys** are used to protect PINs sent to a remote location. These keys must be created as sending cross-domain keys.

In addition to a PIN protection key, PIN commands may require a PIN validation key. A PIN validation key generates a PIN by encrypting the validation data as described in step 3 of the “PIN Generation Algorithm” on page 4-1. The same validation key verifies the PIN and must be stored and associated with the customer’s validation data. For example, you could assign a particular PIN validation key to a range of customer numbers.

PIN validation keys and input PIN protection keys are stored in the cross-domain table with a key use of *PIN. Output PIN protection keys are stored in the cross-domain key table with a key use of *SND.

The GENPIN, VFYPIN, and TRNPIN commands all request the validation data or the encrypted PIN as input data. Internally, these are 8-byte values. Externally, you supply each of these values to the PIN commands as 16 hexadecimal characters representing the 8-byte value. For example, suppose an 8-byte validation data has the following hexadecimal value:

```
Hex 010AF3F9F5C6E2C2
```

You would supply the same value in a 16-byte character hexadecimal format:

```
Char 010AF3F9F5C6E2C2
```

For further information on PIN management, see the *4700 Finance Communication System Controller Programming Library, Volume 5: Cryptographic Programming* book.

Chapter 5. AS/400 Cryptographic Support Commands and Call Interfaces

The user's interface to Cryptographic Support consists of a set of control language (CL) commands and the programs they call, the command processing programs (CPPs). Some CL commands may be typed interactively at your work station. Others must be run from a CL program, or their CPPs can be called directly if your program is written in another AS/400 language.

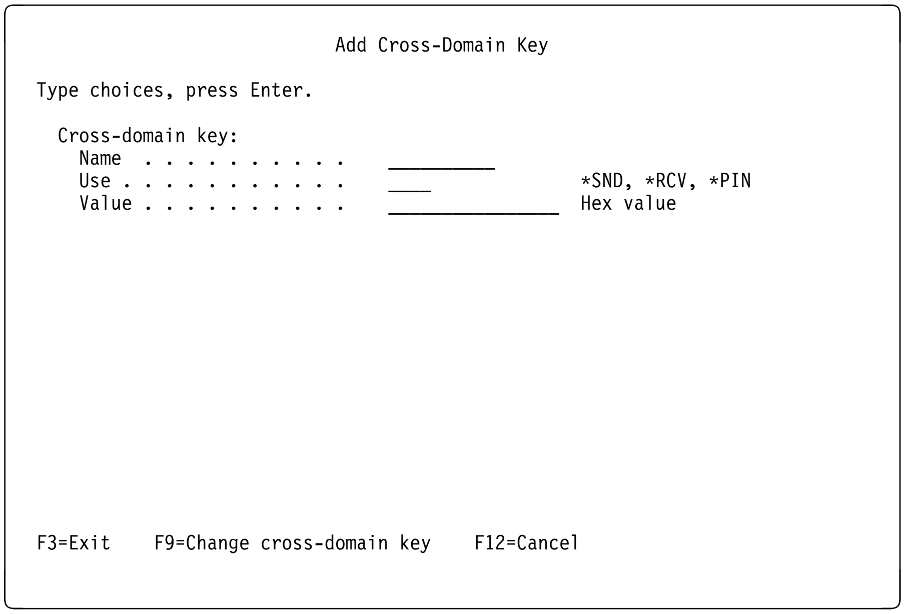
Add Cross-Domain Key (ADDCRSDMNK) Command

The Add Cross-Domain Key (ADDCRSDMNK) command allows you to specify a new cross-domain key that is added to the cross-domain key table. The ADDCRSDMNK command asks for the name of the key, its use, and key value.

Following is the syntax for the ADDCRSDMNK command:

```
ADDCRSDMNK──────────────────────────────────────────────────────────────

                                                    Job: I  Pgm: I
```

When you run the ADDCRSDMNK command, the following display appears:



```
                               Add Cross-Domain Key

 Type choices, press Enter.

   Cross-domain key:
     Name  . . . . . . . . . .    __________
     Use . . . . . . . . . . .    ____               *SND, *RCV, *PIN
     Value . . . . . . . . . .    ________________   Hex value











 F3=Exit    F9=Change cross-domain key    F12=Cancel
```

The prompts in the Add Cross-Domain Key display have the following meaning:

Name. Specify a valid AS/400 name for the cross-domain key. Use this name to refer to the cross-domain key.

Use. Specify *SND for a sending cross-domain key, *RCV for a receiving cross-domain key, or *PIN for a PIN key.

Value. Specify 16 hexadecimal characters that represent the 8-byte value of the cross-domain key. The value specified must have odd parity in every byte.

The following escape messages can be generated for the ADDCRSDMNK command:

CRP0001	Cryptographic Support not found.
CRP0003	Host master key damaged or not defined.
CRP0004	Error occurred while opening file QACRKTBL in QUSRSYS.
CRP0005	Error occurred while processing file QACRKTBL in QUSRSYS.
CRP0008	Error occurred while closing file QACRKTBL in QUSRSYS.

Change Master Key (CHGMSTK) Command

The Change Master Key (CHGMSTK) command installs a new host master key on your system, then decrypts and encrypts the values again in the cross-domain key table under the new host master key. The exclusive-OR operation used on two key parts that you supply produces the new host master key, which must have odd parity in each byte.

To install the new host master key, the value of the old host master key verification code must also be supplied. After installing a new host master key, a message at the bottom of the display shows the new verification code. This verification code should be recorded and stored in a secure place. The CHGMSTK and Verify Master Key (VFYMSTK) commands require this verification code.

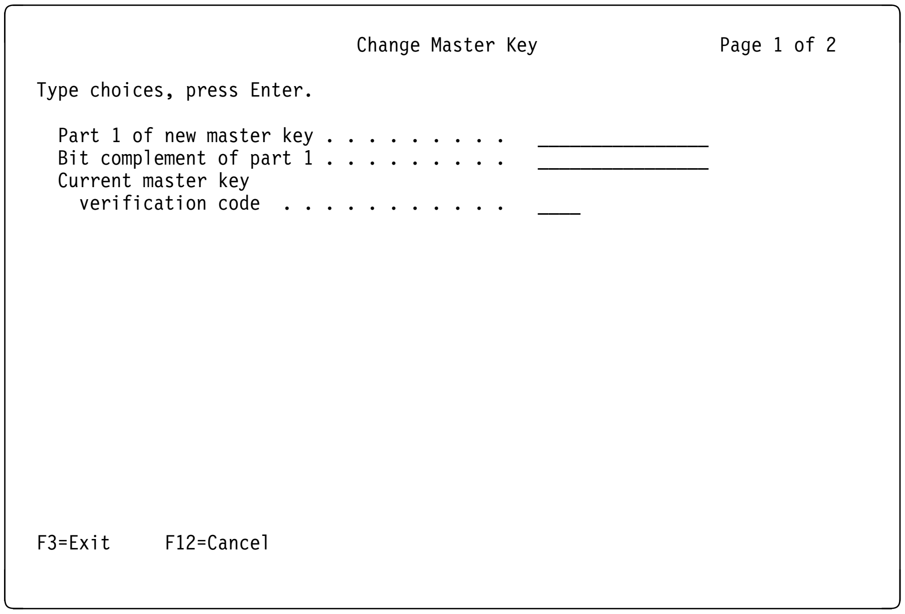
A message sent to the history log, QHST, records the time, date, and job that changed the value of the host master key.

To use the CHGMSTK command, place your system in a restricted state. To reach a restricted state, enter the End System (ENDSYS) command. After installing the master key, return your system to a normal state by restarting your subsystems.

Following is the syntax for the CHGMSTK command:

```
CHGMSTK──────────────────────────────────────────────────────

                                         Job: I  Pgm: I
```

When you run the CHGMSTK command, the following display appears:



```
                              Change Master Key                Page 1 of 2

Type choices, press Enter.

  Part 1 of new master key . . . . . . . . .   ________________
  Bit complement of part 1 . . . . . . . . .   ________________
  Current master key
    verification code  . . . . . . . . . . .   ____










F3=Exit     F12=Cancel
```

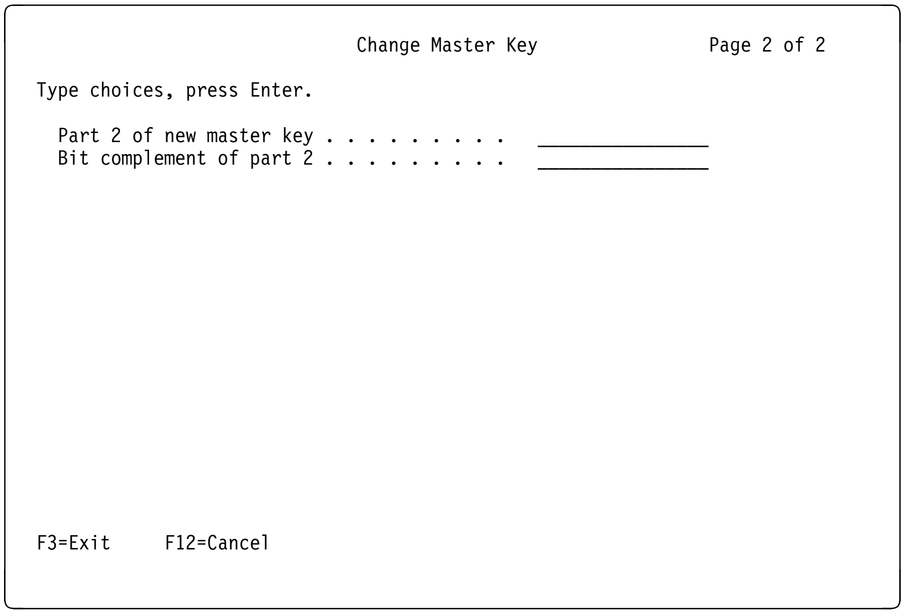
The prompts in the Change Master Key display have the following meaning:

Part 1 of new master key. Specify 16 hexadecimal characters that represent the 8-byte value of the first key part. An exclusive-OR operation of key parts 1 and 2 produces the host master key. The master key must have odd parity in every byte.

Bit complement of part 1. Specify 16 hexadecimal characters that represent the 8-byte complement of part 1. An exclusive-OR operation of key part 1 and its complement should produce 8 bytes of hex FF.

Current master key verification code. Specify 4 hexadecimal characters that represent the 2-byte value of the current master key verification code.

The following is the second display produced by the CHGMSTK command.



```
                              Change Master Key              Page 2 of 2

Type choices, press Enter.

  Part 2 of new master key . . . . . . . . .    ________________
  Bit complement of part 2 . . . . . . . . .    ________________

F3=Exit     F12=Cancel
```

The prompts in the second Change Master Key display have the following meaning:

Part 2 of new master key. Specify 16 hexadecimal characters that represent the 8-byte value of the second key part. An exclusive-OR operation of key parts 1 and 2 produces the host master key. The host master key must have odd parity in every byte.

Bit complement of part 2. Specify 16 hexadecimal characters that represent the 8-byte complement of part 2. An exclusive-OR operation of key part 2 and its complement should produce 8 bytes of hex FF.

The following escape messages can be generated for the CHGMSTK command:

CRP0001	Cryptographic Support not found.
CRP0003	Host master key damaged or not defined.
CRP0004	Error occurred while opening file QACRKTBL in QUSRSYS.
CRP0005	Error occurred while processing file QACRKTBL in QUSRSYS.
CRP0008	Error occurred while closing file QACRKTBL in QUSRSYS.
CRP0201	The system is not in a restricted state.
CRP0208	Master key value has bad parity.
CRP0209	Master key value is a weak key value.

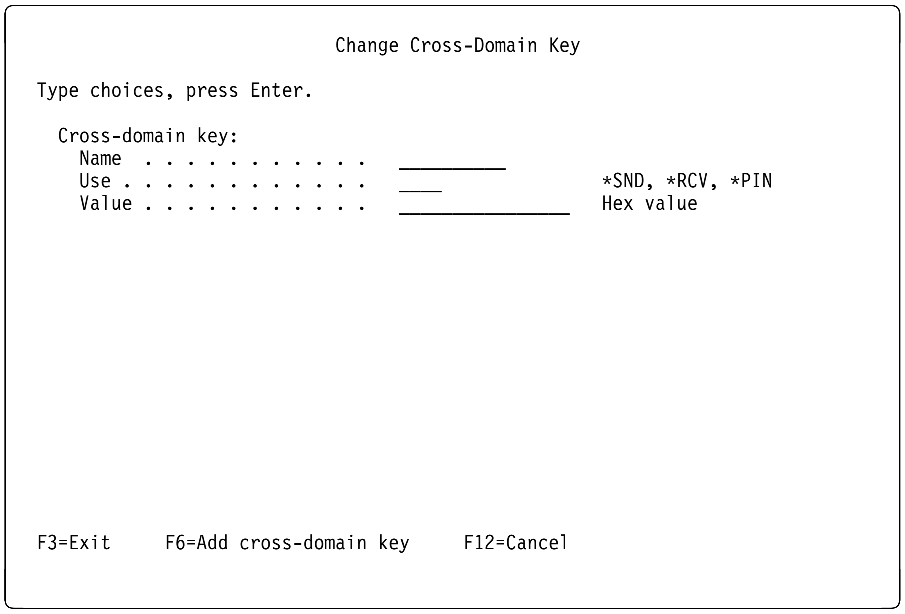
Change Cross-Domain Key (CHGCRSDMNK) Command

The Change Cross-Domain Key (CHGCRSDMNK) command allows you to change the value of a cross-domain key in the key table. The CHGCRSDMNK command asks for the name of the key, its use, and the new key value.

Following is the syntax for the CHGCRSDMNK command:

```
CHGCRSDMNK──────────────────────────────────────────────────────────────
                                                        Job: I  Pgm: I
```

When you run the CHGCRSDMNK command, the following display appears:



```
                            Change Cross-Domain Key

 Type choices, press Enter.

   Cross-domain key:
     Name  . . . . . . . . . . .   __________
     Use . . . . . . . . . . . .   ____             *SND, *RCV, *PIN
     Value . . . . . . . . . . .   ________________   Hex value










 F3=Exit     F6=Add cross-domain key     F12=Cancel
```

The prompts in the Change Cross-Domain Key display have the following meaning:

Name. Specify a valid AS/400 name for the cross-domain key. Use this name to refer to the cross-domain key.

Use. Specify *SND for a sending cross-domain key, *RCV for a receiving cross-domain key, or *PIN for a PIN key.

Value. Specify 16 hexadecimal characters that represent the new 8-byte value of the cross-domain key. This value must have odd parity in every byte.

The following escape messages can be generated for the CHGCRSDMNK command:

CRP0001	Cryptographic Support not found.
CRP0003	Host master key damaged or not defined.
CRP0004	Error occurred while opening file QACRKTBL in QUSRSYS.
CRP0005	Error occurred while processing file QACRKTBL in QUSRSYS.
CRP0008	Error occurred while closing file QACRKTBL in QUSRSYS.

Cipher Data (CPHDTA) Command

Use the Cipher Data (CPHDTA) command to encrypt or decrypt a variable length of data.

Following is the syntax for the CPHDTA command:

```
CPHDTA──DATA input─data─────────────────────────────────────────────────►

►──DTALEN data─length──CPHK cipher─key──RTNVAR return─variable─────────►

          ┌─*ECPH─┐
►──OPTION─┼─*SCPH─┼─────────────────────────────────────────────────────►
          └─*DCPH─┘
                                                                Required
────────────────────────────────────────────────────────────────────────
                                                                Optional
         ┌►*CLEAR──┐        ┌►*NO──┐
►──KTYPE─┤         ├──CHAIN─┤      ├────────────────────────────────────►
         └─*CIPHER─┘        └─*YES─┘

       ┌►*NONE─────────────────┐
►──ICV─┤                       ├────────────────────────────────────────►
       └─initial─chaining─value┘

       ┌►*NO──┐           ┌►X'00'─────────┐
►──PAD─┤      ├──PADCHAR──┤               ├─────────────────────────────►
       └─*YES─┘           └─pad─character─┘

►──RTNDTALEN return─data─length─variable────────────────────────────────

                                                     Pgm: B,I
```

The CPHDTA command uses the following parameters:

DATA
Specifies the data, or a variable containing the data, to be encrypted or decrypted. The data must be at least as long as the length specified in the DTALEN parameter. If you indicate specific encryption (*SCPH) in the OPTION parameter, only values 0 through 9 and A through F may be specified in this parameter.

DTALEN
Specifies the length, or a variable containing the length, of the data to be encrypted or decrypted.

When encrypting data, the length must be a multiple of 8 bytes if both the PAD and CHAIN parameters are specified as *NO. The length must be less than

65 528 bytes if the PAD parameter is specified as *YES. The length must be less than or equal to 65 528 bytes if the PAD parameter is specified as *NO. If you indicate specific encryption (*SCPH) in the OPTION parameter, this value must be 16.

When decrypting data, the length must be a multiple of 8 bytes unless cipher block chaining was used when the data was encrypted. The length must be less than or equal to 65 528 bytes.

CPHK
Specifies an 8-byte value, or a variable containing an 8-byte value, to be used as the key for the Data Encryption Algorithm (DEA). There are no restrictions on the value of this parameter.

RTNVAR
Specifies a variable to receive the results of the encryption or decryption operation. If PAD is specified as *YES and OPTION is specified as *ECPH, RTNVAR must be at least as long as the next 8-byte multiple past the value specified for DTALEN. Otherwise, RTNVAR must be at least as long as the DTALEN value.

OPTION
Specifies the function to be performed:

***ECPH**. A copy of the data in the DATA parameter is encrypted and placed in the RTNVAR parameter.

***SCPH**. The DATA parameter contains 16 hexadecimal characters that represent 8 bytes of data to encrypt (for example, 'C1' represents 'A'). A copy of the 16 hexadecimal characters in the DATA parameter is converted to an 8-byte field and encrypted. The resulting 8 bytes of ciphertext are converted back to 16 hexadecimal characters and placed in the RTNVAR parameter.

***DCPH**. A copy of the data in the DATA parameter is decrypted and placed in the variable specified in the RTNVAR parameter.

KTYPE
Specifies the format of the key specified in the CPHK parameter:

***CLEAR**. The cipher key is specified in plaintext. If the OPTION parameter is *SCPH, KTYPE must specify *CLEAR.

***CIPHER**. The cipher key is encrypted under the host master key. If the CHAIN or PAD option is specified as *YES, KTYPE must specify *CIPHER.

CHAIN
Specifies if cipher block chaining is to be used during the cipher operation:

***NO**. Cipher block chaining is not used. If the OPTION parameter is *SCPH or the KTYPE parameter is *CLEAR, CHAIN must specify *NO.

***YES**. Cipher block chaining is used.

ICV
Specifies an 8-byte value, or a variable containing an 8-byte value, to be used as the initial chaining value when performing cipher block chaining. This parameter is ignored if the CHAIN option is *NO.

***NONE**. There is no initial chaining value.

initial-chaining-value. Enter the value to be used as the initial chaining value for cipher block chaining. There are no restrictions on the value for this parameter. A value is required if the CHAIN option is specified *YES.

PAD
Specifies whether padding is to be performed:

<u>*NO</u>. Padding is not performed. If the OPTION parameter is *SCPH, PAD must be *NO.

***YES**. Before encrypting, Cryptographic Support pads DATA out to the next 8-byte multiple using the PADCHAR parameter. A count of the number of pad characters then replaces the last byte.

After decrypting DATA, Cryptographic Support leaves the pad characters and count byte appended to the data. The length of the data without the pad characters is returned in the variable RTNDTALEN.

PADCHAR
Specifies a 1-byte value, or a variable containing a 1-byte value, to be used as the pad character when the OPTION parameter is *ECPH and the PAD parameter is *YES. The default is hex 00.

RTNDTALEN
Specifies a variable to receive the length of the returned data. When the OPTION parameter is *DCPH and the PAD parameter is *YES, this length is the length of the decrypted data without padding. Otherwise, RTNDTALEN contains the length of the data placed in the RTNVAR parameter.

See “Special Considerations” on page 5-9 for additional information on using the CPHDTA parameters. See Chapter 8, “Coding Examples,” for examples of using the CPHDTA command.

CPHDTA CPP (QCRCIPHR)

The program QCRCIPHR, in library QCRP, can access the same cryptographic functions as the CPHDTA command. QCRCIPHR expects 5 or 11 parameters to be passed on the call.

The parameter types and their required order are listed below. The following five parameters are required:

Order	Required Parameter	Type	Length (Bytes)
1	DATA	Variable or constant.	variable
2	DTALEN	Packed variable or constant with 5 digits, 0 decimal places.	
3	CPHK	Variable or constant.	8
4	RTNVAR	Variable.	variable, at least 8
5	OPTION	Variable or constant. Valid values are E (encryption), S (specific encryption), and D (decryption).	1

The following six parameters are optional. However, if one is specified, all must be specified.

Order	Optional Parameter	Type	Length (Bytes)
6	KTYPE	Variable or constant. Valid values are C to specify a clear key type, and E, to specify an encrypted key type.	1
7	CHAIN	Variable or constant. Valid values are N, to specify no chaining is to be used, and Y, to specify chaining is to be used.	1
8	ICV	Variable or constant. Specify 8 bytes of hex zero for no initial chaining value.	8
9	PAD	Variable or constant. Valid values are N, to specify no padding is to be used, and Y, to specify padding is to be used.	1
10	PADCHAR	Variable or constant.	1
11	RTNDTALEN	Packed variable with 5 digits, 0 decimal places.	

See Chapter 8, “Coding Examples,” for examples of calling QCRCIPHR.

Special Considerations

Consider the following information when writing a program that uses the CPHDTA functions:

- When the RTNVAR parameter is longer than the data being returned, the value returned is left-adjusted, and the remainder of RTNVAR is unchanged.
- To overlay the input to the CPHDTA command with the output, specify the same area for the DATA and RTNVAR parameters. Be sure the language in which your application is written can support this overlay.
- Parameters containing ciphertext (the DATA parameter when decrypting and the RTNVAR parameter when encrypting) can be any binary value and should always be treated as character variables in your program.
- Parameters containing plaintext (the DATA parameter when encrypting and the RTNVAR parameter when decrypting) may be of any data type. If your application uses character fields for the plaintext parameters containing decimal data, some adjustments may be required to maintain the sign and decimal position when moving data between character and decimal fields.
- The RTNVAR parameter must be large enough to contain the returned data. If it is not, results that cannot be predicted, such as loss of program data, occur.
- Variable length variables, such as those used in BASIC, must be passed in a fixed length format. The variables must not contain the field length, but only the actual data.

The following escape messages can be generated for the CPHDTA command:

CRP0001	Cryptographic Support not found.
CRP0002	Input parameters contain errors.
CRP0003	Host master key damaged or not defined.
CRP0050	DTALEN &1 exceeds length of DATA or RTNVAR parameter.
CRP0051	DATA parameter contains characters that are not valid.
CRP0052	DTALEN value contains characters that are not valid.
CRP0053	RTNVAR parameter overlaps end of DATA parameter limits.

Encrypt Cipher Key (ENCCPHK) Command

The Encrypt Cipher Key (ENCCPHK) command encrypts a plaintext data encrypting key under the host master key.

Following is the syntax for the ENCCPHK command:

```
                                                               Required

ENCCPHK──CLRK cleartext─data─encrypting─key──────────────────────────────►

►──CPHK cipher─key─return─variable───────────────────────────────────────
                                                    ┌───────────────────┐
                                                    │Pgm: B,I           │
                                                    └───────────────────┘
```

The ENCCPHK command uses the following parameters:

CLRK
Specifies an 8-byte value, or a variable containing an 8-byte value, of a plaintext data encrypting key. There are no restrictions on the value of this parameter.

CPHK
Specifies an 8-byte variable to receive the value of the data encrypting key specified in the CLRK parameter after it is encrypted under the master key.

ENCCPHK CPP (QCRENCKY)

The program QCRENCKY, in library QCRP, can access the same cryptographic functions as the ENCCPHK command. QCRENCKY expects two parameters to be passed on the call.

The parameter types and their required order are listed below:

Order	Parameter	Type	Length (Bytes)
1	CLRK	Variable or constant	8
2	CPHK	Variable	8

The following escape messages can be generated for the ENCCPHK command:

CRP0001	Cryptographic Support not found.
CRP0003	Host master key damaged or not defined.

Encipher from Master Key (ENCFRMMSTK) Command

The Encipher from Master Key (ENCFRMMSTK) command decrypts a data encrypting key from encryption under the host master key and encrypts it under one or two sending cross-domain keys.

Following is the syntax for the ENCFRMMSTK command:

```
ENCFRMMSTK──CPHK cipher─key──────────────────────────────────────────►

►──CRSDMNK1 sending─cross─domain─key─name────────────────────────────►

►──KRTNVAR1 return─variable──────────────────────────────────────────►
                                                              Required
──────────────────────────────────────────────────────────────────────
                                                              Optional
              ┌►*NONE────────────────────────┐
►──CRSDMNK2───┤                              ├───────────────────────►
              └─sending─cross─domain─key─name┘

►──KRTNVAR2 return─variable─────────────────────────────────────────

                                                  ┌─────────┐
                                                  │Pgm: B,I │
                                                  └─────────┘
```

The ENCFRMMSTK command uses the following parameters:

CPHK
Specifies an 8-byte value, or a variable containing an 8-byte value, that is the value of a data encrypting key encrypted under the host master key. There are no restrictions on the value of this parameter.

CRSDMNK1
Specifies the name, or a variable containing the name, of a sending cross-domain key. The value in the CPHK parameter is decrypted using the host master key and encrypted using this sending cross-domain key.

KRTNVAR1
Specifies an 8-byte variable to receive the data encrypting key encrypted under the sending cross-domain key.

CRSDMNK2
Specifies the name of a second sending cross-domain key. The value in the CPHK parameter is decrypted using the host master key and encrypted using this second sending cross-domain key.

***NONE**. No sending cross-domain key is to be used to encrypt the data encrypting key.

sending-cross-domain-key-name. Enter the name, or a variable containing the name, of a cross-domain key.

KRTNVAR2
Specifies an 8-byte variable to receive the data encrypting key encrypted under the second sending cross-domain key.

ENCFRMMSTK CPP (QCRENCFR)

The program QCRENCFR, in library QCRP, can access the same cryptographic functions as the ENCFRMMSTK command. QCRENCFR requires three parameters to be passed on the call. An additional two parameters are optional.

The parameter types and their required order are listed below. The following three parameters are required:

Order	Parameter	Type	Length (Bytes)
1	CPHK	Variable or constant	8
2	CRSDMNK1	Variable or constant	10
3	KRTNVAR1	Variable	8

The following two parameters are optional. However, if either is specified, both must be specified.

Order	Parameter	Type	Length (Bytes)
4	CRSDMNK2	Variable or constant	10
5	KRTNVAR2	Variable	8

The following escape messages can be generated for the ENCFRMMSTK command:

CRP0001	Cryptographic Support not found.
CRP0002	Input parameters contain errors.
CRP0003	Host master key damaged or not defined.
CRP0004	Error occurred while opening file QACRKTBL in QUSRSYS.
CRP0005	Error occurred while processing file QACRKTBL in QUSRSYS.
CRP0006	Key &1 with use &2 damaged.
CRP0008	Error occurred while closing file QACRKTBL in QUSRSYS.

CRP0701 Number of parameters not valid.

Encipher to Master Key (ENCTOMSTK) Command

The Encipher to Master Key (ENCTOMSTK) command decrypts a data encrypting key from encryption under a receiving cross-domain key and encrypts it under the host master key.

Following is the syntax for the ENCTOMSTK command:

```
                                                                  Required
ENCTOMSTK──CPHK cipher─key─return─variable────────────────────────────────►

►──CRSDMNK receiving─cross─domain─key─name────────────────────────────────►

►──ENCCPHK encrypted─cipher─key──────────────────────────────────────────

                                                          Pgm: B,I
```

The ENCTOMSTK command uses the following parameters:

CPHK
Specifies an 8-byte variable to receive the data encrypting key encrypted under the host master key.

CRSDMNK
Specifies the name, or a variable containing the name, of a receiving cross-domain key. The value in the ENCCPHK parameter is decrypted using the receiving cross-domain key and encrypted using the host master key.

ENCCPHK
Specifies the 8-byte value, or a variable containing the 8-byte value, of a data encrypting key encrypted under the receiving cross-domain key specified in the CRSDMNK parameter.

ENCTOMSTK CPP (QCRENCTO)

The program QCRENCTO, in library QCRP, can access the same cryptographic functions as the ENCTOMSTK command. QCRENCTO requires three parameters to be passed on the call.

The parameter types and their required order are listed below:

Order	Parameter	Type	Length (Bytes)
1	CPHK	Variable	8
2	CRSDMNK	Variable or constant	10
3	ENCCPHK	Variable or constant	8

The following escape messages can be generated for the ENCTOMSTK command:

CRP0001 Cryptographic Support not found.

CRP0003 Host master key damaged or not defined.

CRP0004	Error occurred while opening file QACRKTBL in QUSRSYS.
CRP0005	Error occurred while processing file QACRKTBL in QUSRSYS.
CRP0006	Key &1 with use &2 damaged.
CRP0007	Key &1 with use &2 not found in key table.
CRP0008	Error occurred while closing file QACRKTBL in QUSRSYS.

Generate Cipher Key (GENCPHK) Command

The Generate Cipher Key (GENCPHK) command generates a pseudo-random data encrypting key encrypted under the host master key. The GENCPHK command optionally returns the random data encrypting key encrypted under a sending or receiving cross-domain key.

Following is the syntax for the GENCPHK command:

```
GENCPHK──CPHK cipher─key─return─variable──────────────────────────────────►
                                                                   Required
───────────────────────────────────────────────────────────────────────────
                                                                   Optional
               ┌►*NONE──────────────────────────┐
►──SNDCRSDMNK──┤                                ├─────────────────────────►
               └─sending─cross─domain─key─name──┘

►──SNDRTNVAR return─variable──────────────────────────────────────────────►

               ┌►*NONE────────────────────────────┐
►──RCVCRSDMNK──┤                                  ├───────────────────────►
               └─receiving─cross─domain─key─name──┘

►──RCVRTNVAR return─variable───────────────────────────────────────────────

                                                       ┌──────────────┐
                                                       │Pgm: B,I      │
                                                       └──────────────┘
```

The GENCPHK command uses the following parameters:

CPHK
Specifies an 8-byte variable to receive the value of the generated data encrypting key encrypted under the host master key.

SNDCRSDMNK
Specifies the name of a sending cross-domain key that is to be used to encrypt the generated data encrypting key:

***NONE**. No sending cross-domain key is to be used to encrypt the generated data encrypting key.

sending-cross-domain-key-name. Enter the name, or a variable containing the name, of a sending cross-domain key.

SNDRTNVAR
Specifies an 8-byte variable to receive the value of the generated data encrypting key encrypted under the sending cross-domain key specified in the SNDCRSDMNK parameter.

RCVCRSDMNK
Specifies the name of a receiving cross-domain key that is to be used to encrypt the generated data encrypting key:

***NONE**. No receiving cross-domain key is to be used to encrypt the generated data encrypting key.

receiving-cross-domain-key-name. Enter the name, or a variable containing the name, of a receiving cross-domain key.

RCVRTNVAR
Specifies an 8-byte variable to receive the value of the generated data encrypting key encrypted under the receiving cross-domain key specified in the RCVCRSDMNK parameter.

GENCPHK CPP (QCRGENKY)

The program QCRGENKY, in library QCRP, can access the same cryptographic functions as the GENCPHK command QCRGENKY requires one, three, or five parameters to be passed on the call.

The following parameter is required:

Order	Parameter	Type	Length (Bytes)
1	CPHK	Variable	8

The following four parameters are optional. However, either parameters 2 and 3 must be specified, or all five parameters must be specified.

Order	Parameter	Type	Length (Bytes)
2	SNDCRSDMNK	Variable or constant	10
3	SNDRTNVAR	Variable	8
4	RCVCRSDMNK	Variable or constant	10
5	RCVRTNVAR	Variable	8

The following escape messages can be generated for the GENCPHK command:

CRP0001	Cryptographic Support not found.
CRP0003	Host master key damaged or not defined.
CRP0004	Error occurred while opening file QACRKTBL in QUSRSYS.
CRP0005	Error occurred while processing file QACRKTBL in QUSRSYS.
CRP0008	Error occurred while closing file QACRKTBL in QUSRSYS.
CRP0901	Number of parameters not valid.
CRP0904	Cannot encipher generated key under some cross-domain keys.

Generate Message Authentication Code (GENMAC) Command

The Generate Message Authentication Code (GENMAC) command encrypts a variable length of data using cipher block chaining and returns the last 8 bytes as a message authentication code.

Following is the syntax for the GENMAC command:

```
GENMAC──DATA input─data──────────────────────────────────────────────►

►──DTALEN data─length──CPHK cipher─key───────────────────────────────►

►──ICV initial─chaining─value──RTNVAR return─variable────────────────►
                                                              Required
──────────────────────────────────────────────────────────────────────
                                                              Optional
        ┌►*NO──┐           ┌►X'00'─────────┐
►──PAD──┤      ├──PADCHAR──┤               ├─────────────────────────
        └─*YES─┘           └─pad─character─┘

                                                   ┌──────────────┐
                                                   │Pgm: B,I      │
                                                   └──────────────┘
```

The GENMAC command uses the following parameters:

DATA
Specifies the data, or a variable containing the data, to be encrypted. The DATA must be at least as long as the length specified in the DTALEN parameter.

DTALEN
Specifies the length, or a variable containing the length, of the data to be encrypted. The length must be less than 32 760 bytes.

CPHK
Specifies an 8-byte value, or a variable containing an 8-byte value, to be used as the key for the DEA. This value must be the value of the key encrypted under the host master key. There are no restrictions on the value of this parameter.

ICV
Specifies an 8-byte value, or a variable containing an 8-byte value, to be used as the initial chaining value when performing cipher block chaining. There are no restrictions on the value of this parameter.

RTNVAR
Specifies a variable to receive the 8-byte message authentication code.

PAD
Specifies whether padding is to be performed by Cryptographic Support:

<u>*NO</u>. Padding is not performed.

***YES**. Before encrypting, Cryptographic Support pads DATA out to the next 8-byte multiple using the PADCHAR parameter. The last byte is then replaced with a count of the number of pad characters.

PADCHAR
Specifies a 1-byte value, or a variable containing a 1-byte value, to be used as the pad character when the PAD parameter is *YES. The default is hex 00.

GENMAC CPP (QCRGENMA)

The program QCRGENMA, in library QCRP, can access the same cryptographic functions as the GENMAC command. QCRGENMA requires seven parameters to be passed on the call.

The parameter types and their required order are listed below:

Order	Parameter	Type	Length (Bytes)
1	DATA	Variable or constant.	variable
2	DTALEN	Packed variable or constant with 5 digits, 0 decimal places.	
3	CPHK	Variable or constant.	8
4	ICV	Variable or constant. Specify 8 bytes of hex zero for no initial chaining value.	8
5	RTNVAR	Variable.	8
6	PAD	Character variable or constant. Valid values are N to specify no padding is to be used, and Y to specify padding is to be used.	1
7	PADCHAR	Variable or constant.	1

The following escape messages can be generated for the GENMAC command:

CRP0001	Cryptographic Support not found.
CRP0002	Input parameters contain errors.
CRP0003	Host master key damaged or not defined.
CRP0052	DTALEN value contains characters that are not valid.
CRP0055	DTALEN &1 is larger than length of DATA parameter.

Generate PIN (GENPIN) Command

The Generate PIN (GENPIN) command produces a personal identification number (PIN) that is algorithmically related to the customer's validation data. The generated PIN value contains 16 decimal digits that may be assigned to a customer. If the customer has a preselected PIN value, these digits may be used as an intermediate PIN.

Following is the syntax for the GENPIN command:

```
                                                                         Required

GENPIN───PINVLDK PIN─validation─key──────────────────────────────────────────────►

►───DECTBL decimalization─table───VLDDTA validation─data─────────────────────────►

►───PINRTNVAR PIN─return─variable──────────────────────────────────────────────

                                                               Pgm: B,I
```

The GENPIN command uses the following parameters:

PINVLDK
Specifies the name, or a variable containing the name, of a PIN validation key used to encrypt the validation data. The PIN validation key must exist in the cross-domain key table with a defined use of PIN.

DECTBL
Specifies 16 numeric digits (0 through 9), or a character variable containing 16 numeric digits, to be used as the decimalization table when generating the PIN.

VLDDTA
Specifies 16 hexadecimal characters, or a character variable containing 16 hexadecimal characters, that represent the 8 bytes of validation data to be used for generating the PIN.

PINRTNVAR
Specifies a variable to receive the 16-digit generated PIN.

GENPIN CPP (QCRGENPN)

The program QCRGENPN, in library QCRP, can access the same cryptographic functions as the GENPIN command. QCRGENPN expects four parameters to be passed on the call.

The parameter types and their required order are listed below:

Order	Parameter	Type	Length (Bytes)
1	PINVLDK	Variable or constant	10
2	DECTBL	Character variable or constant	16
3	VLDDTA	Character variable or constant	16
4	PINRTNVAR	Variable	16

The following escape messages can be generated for the GENPIN command:

CRP0001	Cryptographic Support not found.
CRP0003	Host master key damaged or not defined.
CRP0004	Error occurred while opening file QACRKTBL in QUSRSYS.
CRP0005	Error occurred while processing file QACRKTBL in QUSRSYS.
CRP0006	Key &1 with use &2 damaged.
CRP0007	Key &1 with use &2 not found in key table.
CRP0008	Error occurred while closing file QACRKTBL in QUSRSYS.
CRP1101	DECTBL value contains characters that are not valid.
CRP1104	VLDDTA value contains characters that are not valid.

Generate Cross-Domain Key (GENCRSDMNK) Command

The Generate Cross-Domain Key (GENCRSDMNK) command generates pseudo-random key values, installs them in the cross-domain key table, and prints a list of the keys generated. The spooled file containing this list is sent to the output queue of the job from which the command is submitted.

Following is the syntax for the GENCRSDMNK command:

```
                                                                    Required
GENCRSDMNK──CRSDMNK key─name──────────────────────────────────────────────────►

►──NBR number─of─keys──SEED seed─value────────────────────────────────────────►

           ┌─*SND───────┐
►──KUSE────┼─*RCV───────┼─────────────────────────────────────────────────────
           └─*PIN───────┘

                                                          Job: B,I  Pgm: B,I
```

The GENCRSDMNK command uses the following parameters:

CRSDMNK
Specifies a valid AS/400 name, or a variable containing a name, to be used when generating the key names.

All generated key names are 10 characters long with numerics in positions 7 through 10. When you specify a key name, any characters in positions 7 through 10 must be numeric. If you specify a key name base that has fewer than 10 characters, the name is filled in with 0's up to the last character, which is made a 1. This becomes the name of the first key generated.

For each following key value, the last 4 characters of the key name, which are always numbers, are incremented by 1 to become the next key name. GENCRSDMNK does not generate any key names past 9999.

The following examples show the name specified for the CRSDMNK parameter and the first key name generated from it:

CRSDMNK Value	Generated Key Name
A	A000000001
ABCDEF12	ABCDEF1201
A123	A123000001
ABCDEF1234	ABCDEF1234

NBR
Specifies the number, or a variable containing the number, of keys to be generated. The value specified plus the 4-digit number from the first key name cannot exceed 10 000 keys.

SEED
Specifies 16 hexadecimal characters, or a character variable containing 16 hexadecimal characters, that represent the 8-byte value to be used as the seed to start the random number generation routine.

KUSE

Specifies the use of the keys to be generated:

***SND**. The generated keys are added to the cross-domain key table as sending cross-domain keys.

***RCV**. The generated keys are added to the cross-domain key table as receiving cross-domain keys.

***PIN**. The generated keys are added to the cross-domain key table as PIN keys.

Special Considerations

The following special considerations apply when you use GENCRSDMNK:

- Because GENCRSDMNK creates a list of the plaintext values of the generated keys, issue the GENCRSDMNK command only in a secure environment. Print the list of the values immediately, and store it in a safe place.
- The name of the print file is QPCRGENX. If this file is spooled, it appears in the default output queue of the job from which the GENCRSDMNK command is submitted.
- If the SAVE attribute of the spooled file is *YES, it is not deleted from the output queue after it has printed. Use option 4 on the Output Queue display to delete the spooled file from the system. To avoid spooling, change the SPOOL parameter of the print file to *NO, using the Change Print File (CHGPRTF) command before running GENCRSDMNK.

The following escape messages can be generated for the GENCRSDMNK command:

CRP0001	Cryptographic Support not found.
CRP0003	Host master key damaged or not defined.
CRP0004	Error occurred while opening file QACRKTBL in QUSRSYS.
CRP0005	Error occurred while processing file QACRKTBL in QUSRSYS.
CRP0008	Error occurred while closing file QACRKTBL in QUSRSYS.
CRP0601	Cross-domain key value &1 is not valid.
CRP0602	A key name cannot be generated past &1.
CRP0603	Error occurred while generating keys.
CRP0605	SEED value hex &1 contains characters that are not valid.
CRP0606	NBR &1 not valid.

Remove Cross-Domain Key (RMVCRSDMNK) Command

The Remove Cross-Domain Key (RMVCRSDMNK) command removes one or more specified keys from the cross-domain key table.

Following is the syntax for the RMVCRSDMNK command:

```
                                                                  Required
                           ┌─*ALL──────────────┐
RMVCRSDMNK───CRSDMNK───────┼─generic*─key─name─┼──────────────────────────────►
                           └─key─name──────────┘

                 ┌─*ALL─┐
►───KUSE─────────┼─*SND─┼──────────────────────────────────────────────────────
                 ├─*RCV─┤
                 └─*PIN─┘

                                                          ┌─────────────────────┐
                                                          │Job: B,I  Pgm: B,I   │
                                                          └─────────────────────┘
```

The RMVCRSDMNK command uses the following parameters:

CRSDMNK
Specifies the name or generic name of the keys, with a key use specified in the KUSE parameter, that are to be removed from the cross-domain key table. This parameter can also specify that all keys defined with the use specified in the KUSE parameter are to be removed.

***ALL**. All keys with a key use specified in the KUSE parameter are to be removed from the cross-domain key table.

generic-key-name*. All keys with the same generic name with a key use specified in the KUSE parameter are to be removed from the cross-domain key table. To specify a generic name, add an asterisk after the characters that are common in all the key names to be removed (for example, ABC*). If an asterisk is not included with the name, the system assumes that the name is a complete key name.

key-name. The key with the specified name with a key use specified in the KUSE parameter is to be removed from the cross-domain key table.

KUSE
Specifies the key use of the keys specified in the CRSDMNK parameter that are to be removed from the cross-domain key table. This parameter can also specify that all key uses of the named keys are to be removed.

***ALL**. The keys named in the CRSDMNK parameter with any key use are to be removed.

***SND**. The keys named in the CRSDMNK parameter with a key use of sending are to be removed.

***RCV**. The keys named in the CRSDMNK parameter with a key use of receiving are to be removed.

***PIN**. The keys named in the CRSDMNK parameter with a key use of PIN are to be removed.

The following escape messages can be generated for the RMVCRSDMNK command:

CRP0004	Error occurred while opening file QACRKTBL in QUSRSYS.
CRP0005	Error occurred while processing file QACRKTBL in QUSRSYS.
CRP0007	Key &1 with use &2 not found in key table.
CRP0008	Error occurred while closing file QACRKTBL in QUSRSYS.
CRP0308	Key &1 with use &2 in use.
CRP0503	Error occurred when removing keys from key table.
CRP0504	Error occurred when clearing key table.

Set Master Key (SETMSTK) Command

The Set Master Key (SETMSTK) command installs a new host master key on your system. If an old master key exists, the values in the cross-domain key table are encrypted again under the new host master key value.

The exclusive-OR operation, used on the two key part values you supply, produces the new host master key, which must have odd parity in each byte.

After installing the new host master key, a message at the bottom of the display shows the verification code. This verification code should be recorded and stored in a secure place. The CHGMSTK and VFYMSTK commands require this verification code.

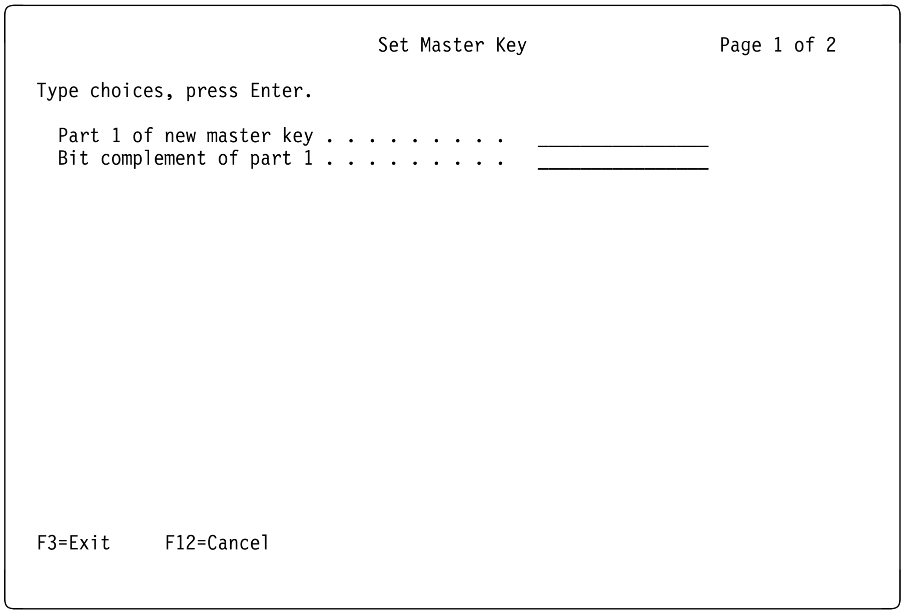
A message is sent to the history log, QHST, to record the time, date, and job that set the new host master key.

To use the SETMSTK command, place your system in a restricted state. To get to a restricted state, enter the End System (ENDSYS) command. After installing the master key, return your system to a normal state by restarting your subsystems.

Following is the syntax for the SETMSTK command:

```
SETMSTK──────────────────────────────────────────────────────────

                                                  Job: I  Pgm: I
```

When you run the SETMSTK command, the following display appears:



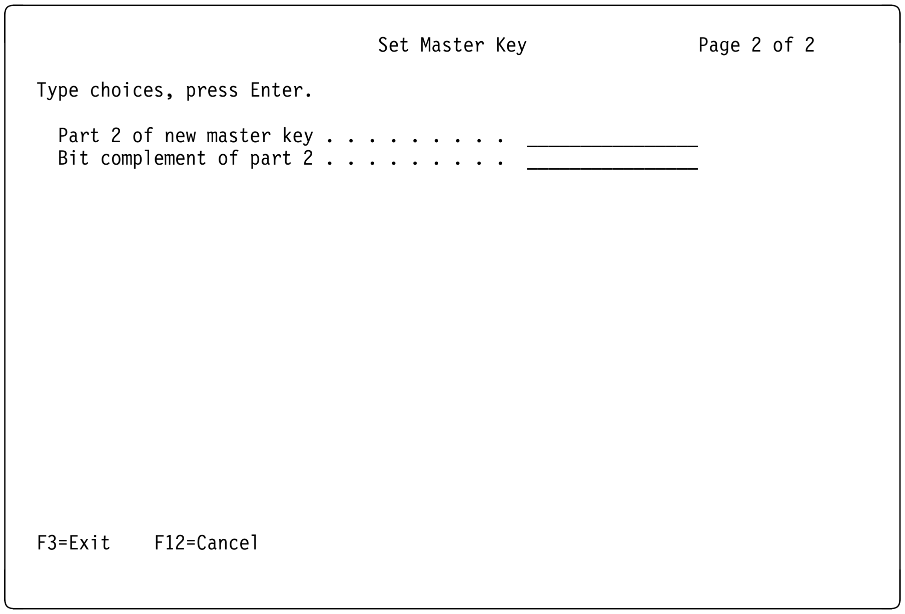
```
                              Set Master Key                Page 1 of 2

 Type choices, press Enter.

   Part 1 of new master key . . . . . . . . .   ________________
   Bit complement of part 1 . . . . . . . . .   ________________




 F3=Exit     F12=Cancel
```

The prompts in the first Set Master Key display have the following meaning:

Part 1 of new master key. Specify 16 hexadecimal characters that represent the 8-byte value of the first key part. An exclusive-OR operation of key parts 1 and 2 produce the host master key. The master key must have odd parity in every byte.

Bit complement of part 1. Specify 16 hexadecimal characters that represent the 8-byte complement of part 1. An exclusive-OR operation of key part 1 and its complement should produce 8 bytes of hex FF.

Following is the second display that appears when you run the SETMSTK command:



```
                              Set Master Key              Page 2 of 2

 Type choices, press Enter.

   Part 2 of new master key . . . . . . . . .  ________________
   Bit complement of part 2 . . . . . . . . .  ________________




 F3=Exit    F12=Cancel
```

The prompts in the second Set Master Key display have the following meaning:

Part 2 of new master key. Specify 16 hexadecimal characters that represent the 8-byte value of the second key part. An exclusive-OR operation of key parts 1 and 2 produce the host master key. The master key must have odd parity in every byte.

Bit complement of part 2. Specify 16 hexadecimal characters that represent the 8-byte complement of part 2. An exclusive-OR operation of key part 2 and its complement should produce 8 bytes of hex FF.

The following escape messages can be generated for the SETMSTK command:

CRP0001	Cryptographic Support not found.
CRP0004	Error occurred while opening file QACRKTBL in QUSRSYS.
CRP0005	Error occurred while processing file QACRKTBL in QUSRSYS.
CRP0008	Error occurred while closing file QACRKTBL in QUSRSYS.
CRP0201	The system is not in a restricted state.
CRP0208	Master key value has bad parity.
CRP0209	Master key value is a weak key value.

Translate PIN (TRNPIN) Command

The Translate PIN (TRNPIN) command decrypts a PIN from encryption under an input PIN protection key and encrypts it under an output PIN protection key.

Following is the syntax for the TRNPIN command:

```
                                                                  Required
TRNPIN──INPINPTCK input─PIN─protection─key──────────────────────────────►

►──ENCPIN enciphered─PIN───OUTPINPTCK output─PIN─protection─key────────►

►──RTNVAR return─variable──────────────────────────────────────────────

                                                    ┌─────────────┐
                                                    │Pgm: B,I     │
                                                    └─────────────┘
```

The TRNPIN command uses the following parameters:

INPINPTCK
Specifies the name, or a variable containing the name, of the input PIN protection key under which the PIN is encrypted. This key must exist in the cross-domain key table with a defined use of PIN.

ENCPIN
Specifies 16 hexadecimal characters, or a character variable containing 16 hexadecimal characters, that represent a PIN encrypted under the input PIN protection key.

OUTPINPTCK
Specifies the name, or a variable containing the name, of the output PIN protection key. The value specified in the ENCPIN parameter is decrypted using the input PIN protection key and encrypted using the output PIN protection key.

This key must exist in the cross-domain key table with a defined use of sending.

RTNVAR
Specifies a variable to receive the 16-character value of the PIN encrypted under the output PIN protection key.

TRNPIN CPP (QCRTRNPN)

The program QCRTRNPN, in library QCRP, can access the same cryptographic functions as the TRNPIN command. QCRTRNPN requires four parameters to be passed on the call.

The parameter types and their required order are listed below:

Order	Parameter	Type	Length (Bytes)
1	INPINPTCK	Variable or constant	10
2	ENCPIN	Character variable or constant	16
3	OUTPINPTCK	Variable or constant	10
4	RTNVAR	Variable	16

The following escape messages can be generated for the TRNPIN command:

CRP0001	Cryptographic Support not found.
CRP0003	Host master key damaged or not defined.
CRP0004	Error occurred while opening file QACRKTBL in QUSRSYS.
CRP0005	Error occurred while processing file QACRKTBL in QUSRSYS.
CRP0006	Key &1 with use &2 damaged.
CRP0007	Key &1 with use &2 not found in key table.
CRP0008	Error occurred while closing file QACRKTBL in QUSRSYS.
CRP1103	ENCPIN value contains characters that are not valid.

Verify Master Key (VFYMSTK) Command

The Verify Master Key (VFYMSTK) command compares the verification code you type in on the prompt display with the verification code of the current host master key. If the verification codes do not match, the master key has either been changed since you received your verification code or the verification code was incorrectly typed. If the verification codes match, a message at the bottom of the display indicates that the typed verification code is correct.

Following is the syntax for the VFYMSTK command:

```
VFYMSTK────────────────────────────────────────────────────────

                                              Job: I  Pgm: I
```

When you run the VFYMSTK command, the following display appears:

```
                              Verify Master Key

Type choices, press Enter.

  Current master key
    verification code . . . . . . . . . . . . . . . . .    ____









F3=Exit     F12=Cancel
```

The prompt for the Verify Master Key display has the following meaning:

Current master key verification code. Specify 4 hexadecimal characters that represent the 2-byte verification code of what should be the current host master key.

The following escape messages can be generated for the VFYMSTK command:

CRP0001	Cryptographic Support not found.
CRP0003	Host master key damaged or not defined.

Verify PIN (VFYPIN) Command

The Verify PIN (VFYPIN) command determines if the customer's personal identification number is algorithmically related to the customer's validation data. The result of the test is returned in a 1-byte variable.

Following is the syntax for the VFYPIN command:

```
VFYPIN──INPINPTCK input─PIN─protection─key──────────────────────────────►

►──ENCPIN enciphered─PIN──PINVLDK PIN─validation─key────────────────────►

►──DECTBL decimalization─table──VLDDTA validation─data──────────────────►

►──PINCHKLEN PIN─check─length──RTNVAR return─variable───────────────────►
                                                                Required
────────────────────────────────────────────────────────────────────────
                                                                Optional
               ┌►*NONE──────────────┐       ┌►*NONE──────┐
►──PINPADCHAR──┤                    ├──OFFSET─┤            ├────────────────
               └PIN─pad─character───┘       └offset─data─┘

                                                      Pgm: B,I
```

The VFYPIN command uses the following parameters:

INPINPTCK
Specifies the name, or a variable containing the name, of the input PIN protection key under which the PIN is encrypted. This key must exist in the cross-domain key table with a defined use of PIN.

ENCPIN
Specifies 16 hexadecimal characters, or a character variable containing 16 hexadecimal characters, that represent a PIN in 3624 format encrypted under the input PIN protection key.

PINVLDK
Specifies the name, or a variable containing the name, of a PIN validation key that is to be used to encrypt the validation data. This key must exist in the cross-domain key table with a defined use of PIN.

DECTBL
Specifies 16 numeric digits (0 through 9), or a character variable containing 16 numeric digits, to be used as the decimalization table when verifying the PIN.

VLDDTA
Specifies 16 hexadecimal characters, or a character variable containing 16 hexadecimal characters, that represent the 8 bytes of validation data to be used for verifying the PIN.

PINCHKLEN
Specifies the number, or a variable containing the number, of PIN digits to be verified.

RTNVAR
Specifies a variable to receive the 1-byte verification status. If the PIN is valid, the status is set to 0. If the PIN is invalid, the status is set to 1.

PINPADCHAR
Specifies the hexadecimal character used to pad the PIN before being encrypted. It is removed from the end of the PIN before verification.

***NONE**. The PIN was not padded before being encrypted.

PIN-pad-character. Enter 1 hexadecimal character, or a variable containing 1 hexadecimal character, that represents the value used to pad the PIN before it was encrypted.

OFFSET
Specifies a numeric value to be added to the intermediate PIN created from the validation data before comparing it with the supplied PIN.

***NONE**. No offset value should be added to the intermediate PIN before comparing it with the supplied PIN. Specifying *NONE for the OFFSET parameter is equivalent to entering a value of zero for the parameter.

offset-data. Enter 16 numeric digits (0 through 9), or a character variable containing 16 numeric digits, that represent the value to be added to the intermediate PIN before comparing it with the supplied PIN.

VFYPIN CPP (QCRVFYPN)

The program QCRVFYPN, in library QCRP, can access the same cryptographic functions as the VFYPIN command. QCRVFYPN requires nine parameters to be passed on the call.

The parameter types and their required order are listed below:

Order	Parameter	Type	Length (Bytes)
1	INPINPTCK	Variable or constant	10
2	ENCPIN	Character variable or constant	16
3	PINVLDK	Variable or constant	10
4	DECTBL	Character variable or constant	16
5	VLDDTA	Character variable or constant	16
6	PINCHKLEN	Packed variable or constant with 5 digits, 0 decimal places	
7	RTNVAR	Variable	1
8	PINPADCHAR	Variable or constant	1
9	OFFSET	Character variable or constant	16

The following escape messages can be generated for the VFYPIN command:

CRP0001	Cryptographic Support not found.
CRP0003	Host master key damaged or not defined.
CRP0004	Error occurred while opening file QACRKTBL in QUSRSYS.
CRP0005	Error occurred while processing file QACRKTBL in QUSRSYS.
CRP0006	Key &1 with use &2 damaged.
CRP0007	Key &1 with use &2 not found in key table.
CRP0008	Error occurred while closing file QACRKTBL in QUSRSYS.
CRP1101	DECTBL value contains characters that are not valid.
CRP1103	ENCPIN value contains characters that are not valid.
CRP1104	VLDDTA value contains characters that are not valid.
CRP1105	PINCHKLEN &1 not valid.
CRP1106	PINPADCHAR &1 not valid.
CRP1107	OFFSET value contains characters that are not valid.

Chapter 6. System/36 Environment Cryptographic Support Call Interfaces

On the System/36, subroutines provided Cryptographic Support for RPG II, COBOL, and assembler application programs. The subroutines SUBR30 for System/36-compatible RPG II and SUBR31 for System/36-compatible COBOL are supported on the AS/400 system.

The AS/400 user's interface to SUBR30 and SUBR31 is identical to that on the System/36, except for some new error messages.

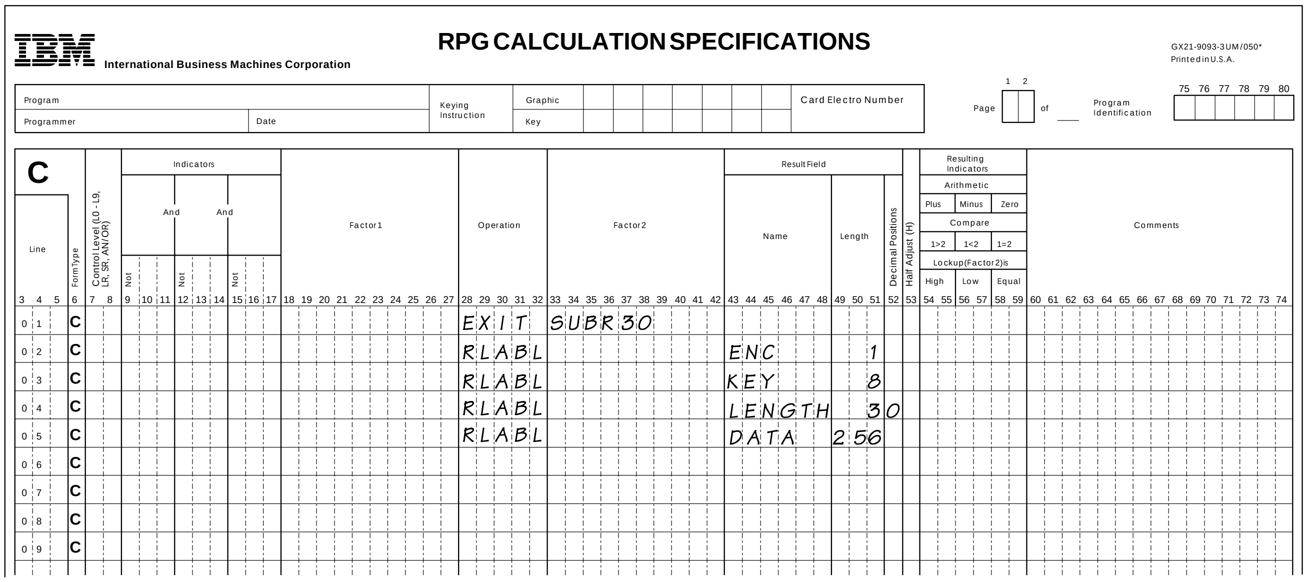
SUBR30 and SUBR31 provide these functions:

- General encryption/decryption
- Specific encryption

Each function has a specific set of parameters.

General Encryption/Decryption

To call the general encryption/decryption function from RPG II, make the following entries on the calculation specifications:



IBM International Business Machines Corporation

RPG CALCULATION SPECIFICATIONS

GX21-9093-3UM/050* Printed in U.S.A.

Line	Form Type	Operation	Factor 2	Result Field Name	Length	Decimal Positions
01	C	EXIT	SUBR30			
02	C	RLABL		ENC	1	
03	C	RLABL		KEY	8	
04	C	RLABL		LENGTH	3	0
05	C	RLABL		DATA	256	
06	C					
07	C					
08	C					
09	C					

RSLQ012

The format of the COBOL call to SUBR31 for general encryption/decryption is:

```
CALL 'SUBR31' USING ENC,KEY,LENGTH,DATA
```

ENC
Is a 1-byte alphameric field that contains an E or a D.

E Means encryption is to be performed.

D Means decryption is to be performed.

KEY
Is an 8-byte alphameric field that contains the key to be used to encrypt or decrypt the data.

LENGTH
Is a 3-byte zoned decimal field that specifies the total length of the data to be encrypted or decrypted. The value specified for LENGTH must be less than or equal to the length of the DATA field. The DEA encrypts and decrypts data in blocks of 8 bytes. Thus, the value of LENGTH must be a multiple of 8 greater than 0. If LENGTH contains a value greater than 256, SUBR30 and SUBR31 assume a value of 256.

DATA
Is an alphameric field that contains the data to be encrypted or decrypted. When SUBR30 or SUBR31 return control to your program, DATA contains the encrypted or decrypted data.

There is no explicit return code field for SUBR30 or SUBR31. However, if an error occurs, SUBR30 and SUBR31 overlay the ENC field with one of the following return codes:

1 The value of ENC is not E, D, or V.

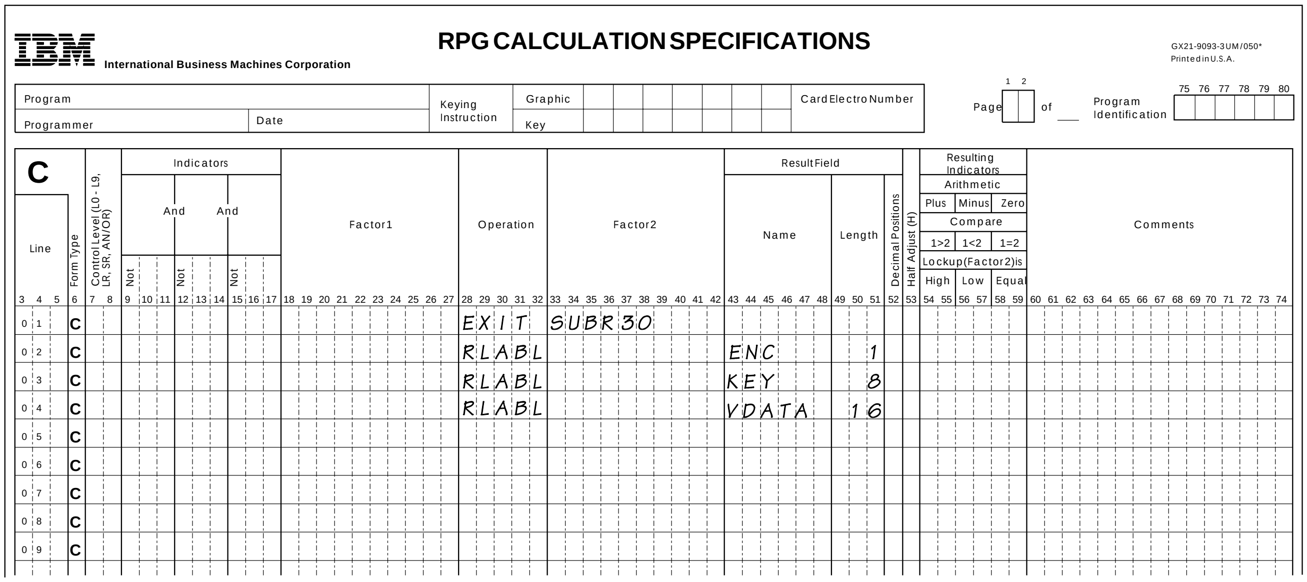
3 The value of LENGTH is not valid.

In addition, some errors may cause exceptions. See “Special Considerations” on page 6-4 for more information.

Specific Encryption

The specific encryption function encrypts character data as if it were hexadecimal data. The input data is considered to be a character representation of hexadecimal data. SUBR30 and SUBR31 convert the input data into its hexadecimal equivalent, encrypt the hexadecimal equivalent, convert the results into a character representation, and pass the character data back to your program.

To call the specific encryption function from RPG II, make the following entries on the calculation specifications:



IBM International Business Machines Corporation

RPG CALCULATION SPECIFICATIONS

GX21-9093-3 UM/050* Printed in U.S.A.

Program		Keying Instruction	Graphic	Card Electro Number
Programmer	Date		Key	

Page of — Program Identification

Line	Form Type	Control Level (L0 - L9, LR, SR, AN/OR)	Indicators	Factor 1	Operation	Factor 2	Result Field Name	Length	Decimal Positions	Half Adjust (H)	Resulting Indicators	Comments
01	C				EXIT	SUBR30						
02	C				RLABL		ENC	1				
03	C				RLABL		KEY	8				
04	C				RLABL		VDATA	16				
05	C											
06	C											
07	C											
08	C											
09	C											

RSLQ013

The format of the COBOL call to SUBR31 for specific encryption is:

```
CALL 'SUBR31' USING ENC,KEY,VDATA
```

ENC
Is a 1-byte alphameric field that contains a V, which indicates that a specific encryption is to be performed.

KEY
Is an 8-byte alphameric field that contains the key to be used to encrypt the data.

VDATA
Is a 16-byte alphameric field that contains the hexadecimal representation (0 through 9 or A through F) of the data to be encrypted. When SUBR30 or SUBR31 returns control to your program, VDATA contains the hexadecimal representation of the encrypted character data. No matter what length is defined for VDATA, SUBR30, and SUBR31, assume that VDATA is a 16-character left-adjusted field.

There is no explicit return code field for SUBR30 and SUBR31. However, if an error occurs, SUBR30 and SUBR31 overlay the ENC field with one of the following return codes:

1 The value of ENC is not E, D, or V.

2 The data in VDATA is not valid data (0 through 9 or A through F).

In addition, some errors may cause exceptions. See “Special Considerations” on page 6-4 for more information.

Special Considerations

Consider the following when using the System/36 environment Cryptographic Support call interfaces:

- SUBR30 and SUBR31 are shipped in library QSSP. To obtain QSSP, the System/36 environment option of the base operating system, 5763-SS1, must be installed.
- SUBR30 and SUBR31 use services provided by the Cryptographic Support/400 licensed program, 5763-CR1. An error occurs if this licensed program is not installed.
- SUBR30 and SUBR31 call QCRP/QCRCIPHR. The user must have authority to QCRP/QCRCIPHR.
- SUBR30 and SUBR31 can be called from any high-level language program as long as the parameters are passed in the defined formats.
- If the value of LENGTH is greater than the defined length of DATA, results that cannot be predicted may occur.

Escape Messages

The following escape messages can be generated by SUBR30 and SUBR31:

CPF2CA0	Cryptographic Support/400 licensed program, 5763-CR1, is not installed.
CPF9820	Not authorized to use library QCRP.
CRP0001	Cryptographic Support not found.

AS/400 System and System/36 Differences

Note the following differences between the AS/400 system and the System/36 when working with the System/36 environment Cryptographic Support call interfaces:

- Besides invalid zoned-decimal data, the following LENGTH problems can cause the ENC field to be set to 3.
  - LENGTH = 0
  - LENGTH not a multiple of 8
- Some errors may cause diagnostic messages to appear on the job log or escape messages.

Chapter 7. Data Security Considerations

Cryptography provides security against external access without permission through wiretapping and theft of removable storage. Access without permission from within the system is protected by controlled use of the cryptographic keys and functions. This can be accomplished by:

- Maintaining the secrecy of values used as input into cryptographic operations
- Limiting authorization to Cryptographic Support

When Cryptographic Support is installed on your system, the commands and their command processing programs (CPPs), as listed in Chapter 5, “AS/400 Cryptographic Support Commands and Call Interfaces,” are owned by the QSECOFR profile and have AUT(*EXCLUDE). To grant other user profiles authority to any commands or to change the public authority to a command, you must also change authority to the CPP.

The System/36 environment cryptographic subroutines SUBR30 and SUBR31 are not shipped with EXCLUDE authority. However, both subroutines call QCRCIPHR, the CPHDTA CPP, to perform the encryption or decryption request. If the user is not authorized to QCRCIPHR, the request will fail.

The cross-domain key table physical file (QUSRSYS/QACRKTBL) is also owned by QSECOFR with AUT(*EXCLUDE). If you grant other users authority to any commands that use the cross-domain key table or refer to a cross-domain key, authority must also be granted to use QACRKTBL. Grant authority to use only the commands needed to perform the command function.

Use system security functions to protect both the data and keys within the system. Restrict use of dedicated service tools (DST), the QSECOFR, and the QSRV and QSRVBAS user IDs by using unique, nontrivial passwords. Restrict the use of the following service and debugging commands to authorized users:

- Dump Job Internal (DMPJOBINT)
- Print Internal Data (PRTINTDTA)
- Trace Internal (TRCINT)
- Dump Job (DMPJOB)
- Start Service Job (STRSRVJOB)
- Trace Job (TRCJOB)
- Dump Object (DMPOBJ)
- Dump System Object (DMPSYSOBJ)
- Start Debug (STRDBG)
- Add Breakpoint (ADDBKP)
- Add Trace (ADDTRC)
- Display Job Log (DSPJOBLOG)
- Start System Service Tools (STRSST)

Chapter 8. Coding Examples

Two methods of accessing Cryptographic Support are diagramed in this chapter. One method uses a control language (CL) program with system commands, and the other calls a command processing program (CPP) using other languages.

Using a CL Program

The following example shows the use of the CPHDTA command within a CL program. Records are read from a file named TRANSX and sensitive portions are encrypted using a key and initial chaining value passed as parameters. The program named SEND is called to send the file across a data link.

The following two fields in the file TRANSX require encryption:

Account number	An 8-byte field in positions 1 through 8
Amount	A 12-byte field in positions 39 through 50

The user requests cipher block chaining with no padding to perform this encryption process.

Note: Not all programming considerations or techniques are shown in the following example. You should review the example before you begin application design and coding.

```
PGM            PARM(&KEY &ICVVAL)
       DCL      VAR(&KEY) TYPE(*CHAR) LEN(8)
       DCL      VAR(&ICVVAL) TYPE(*CHAR) LEN(8)
       DCLF     FILE(TRANSX)
       DCL      VAR(&CPHTRANS) TYPE(*CHAR) LEN(50)
       DCL      VAR(&ACTNO) TYPE(*CHAR) LEN(8)
       DCL      VAR(&AMTNO) TYPE(*CHAR) LEN(12)
LOOP:
                /* READ A RECORD                          */
       RCVF
                /* MONITOR FOR END OF FILE                */
       MONMSG   MSGID(CPF0864) EXEC(GOTO CMDLBL(COMP))

                /* MAKE A COPY OF THE RECORD              */
       CHGVAR   VAR(&CPHTRANS) VALUE(&TRANSX)

                /* CIPHER ACCOUNT FIELD INTO TEMPORARY FIELD*/
       CPHDTA   DATA(%SUBSTRING(&CPHTRANS 1 8)) DTALEN(8) +
                  CPHK(&KEY) RTNVAR(&ACTNO) OPTION(*ECPH) +
                  KTYPE(*CIPHER) CHAIN(*YES) ICV(&ICVVAL)

                /* REPLACE RECORD WITH ENCIPHERED ACCOUNT # */
       CHGVAR   VAR(&SUBSTRING(&CPHTRANS 1 8)) VALUE(&ACTNO)

                /* CIPHER AMOUNT FIELD INTO TEMPORARY FIELD */
        CPHDTA  DATA(%SUBSTRING(&CPHTRANS 39 12)) DTALEN(12) +
                  CPHK(&KEY) RTNVAR(&AMTNO) OPTION(*ECPH) +
                  KTYPE(*CIPHER) CHAIN(*YES) ICV(&ICVVAL)

                /* REPLACE RECORD WITH ENCIPHERED AMOUNT    */
        CHGVAR  VAR(%SUBSTRING(&CPHTRANS 39 12)) VALUE(&AMTNO)

                /* SEND RECORD TO REMOTE LOCATION         */
        CALL    PGM(SEND) PARM(&CPHTRANS)

                /* REPEAT LOOP UNTIL END OF FILE          */
        GOTO    CMDLBL(LOOP)
COMP: ENDPGM
```

Calling QCRCIPHR

The following examples show how to access system Cryptographic Support by calling the program QCRCIPHR in various languages. In each example, it is assumed the calling program reads a record containing a 10-character field, named FLDA, that must be encrypted. The key is passed to the program as an 8-byte variable named KEY.

Cipher block chaining, padding, and an encrypted key value are not required in these examples. You call QCRCIPHR with the shorter parameter list.

Note: Not all programming considerations or techniques are shown in the following examples. You should review the examples before you begin application design and coding.

BASIC

```
  :
25 DECLARE PROGRAM QCRCIPHR (C 16,PD 5,C 8,C 16, C 1) QCRCIPHR.QCRP
  :
85 CALL QCRCIPHR (FLDA$,16,KEY,FLDB$,'E')
95 LET FLDA$ = FLDB$
  :
```

COBOL

```
   :
DATA DIVISION.
   :
WORKING STORAGE SECTION.
01     DATA        PIC X(16).
01     DTALEN      PIC 9(5)        VALUE 16.
01     OPTION      PIC X(1)        VALUE “E”.
   :
LINKAGE SECTION.
01     KEY         PIC X(8).
   :
PROCEDURE DIVISION USING KEY.
   :
MOVE FLDA TO DATA.
CALL “QCRCIPHR” USING DATA, DTALEN, KEY, DATA, OPTION.
MOVE DATA TO FLDA.
   :
```

RPG III

```
                        :
C       *ENTRY        PLIST
C                     PARM           KEY     8
                        :
C                     CALL 'QCRCIPHR'
C                     PARM FLDA      DATA   16
C                     PARM 16        DTALEN  50
C                     PARM           KEY
C       FLDA          PARM           RTNVAR 16
C                     PARM 'E'       OPTION  1
                        :
```

PL/1

```
ENCPROC: PROCEDURE (KEY);
  DECLARE
       KEY            CHAR(8),
       DTALEN         FIXED DEC (5,0) STATIC INIT (16),
       OPTION         CHARACTER(1)    STATIC INIT ('E'),
       DATA           CHARACTER(16);
  DECLARE
       QCRCIPHR ENTRY
                     (CHARACTER (16),
                      FIXED DEC (5,0),
                      CHARACTER (8),
                      CHARACTER (16),
                      CHARACTER (1));
         :
  BEGIN;
         :
       CALL QCRCIPHR(FLDA,DTALEN,KEY,DATA,OPTION);
       FLDA = DATA;
         :
```

Chapter 9. Performance Considerations

Using the Data Encryption Algorithm (DEA) can affect the performance of your system. This should be considered when designing the application program used to access Cryptographic Support.

The following table shows the calculated average amount of time required to encrypt or decrypt varying lengths of data with cipher block chaining on dedicated system models. The first column lists the length of the data processed by one call of QCRCIPHR. The other columns list the amount of time, according to model number.

Data Length in Bytes	**Average Time on System Model (Seconds)**						
	B10	B20	B30	B40	B50	B60	B70
1 024	0.75	0.44	0.63	0.39	0.25	0.14	0.11
4 096	2.98	1.76	2.52	1.58	1.00	0.57	0.42
32 768	23.60	13.95	19.96	12.46	7.91	4.48	3.36

For best performance, encrypt only those fields containing sensitive data. However, when determining which data to encrypt, do not place security standards at risk by placing undue importance on performance objectives.

Glossary

This glossary includes terms and definitions from the *ISO Vocabulary—Information Processing* and the *ISO Vocabulary—Office Machines*, developed by the International Organization for Standardization, Technical Committee 97, Subcommittee 1. Definitions of published segments of the vocabularies are identified by the symbol (I) after the definition; definitions from draft international standards, draft proposals, and working papers in development by the ISO/TC97/SC1 vocabulary subcommittee are identified by the symbol (T) after the definition, indicating final agreement has not yet been reached among participating members.

algorithm. A finite set of well-defined rules for the solution of a problem in a finite number of steps. See also *cryptographic algorithm*.

alphanumeric. Pertaining to the letters, A-Z; numbers, 0-9; and special symbols, $, #, @, ., or _. Synonymous with *alphameric*.

American National Standards Institute (ANSI). An organization sponsored by the Computer and Business Equipment Manufacturers Association for establishing voluntary industry standards.

ANSI. See *American National Standards Institute (ANSI)*.

batch job. A predefined group of processing actions submitted to the system to be performed with little or no interaction between the user and the system. Contrast with *interactive job*.

character. Any letter, number, or other symbol in the data character set that is part of the organization, control, or representation of data.

cipher block chaining. A method of reducing repetitive patterns in ciphertext by performing an exclusive-OR operation on each 8-byte block of data, with the previously encrypted 8-byte block before it is encrypted.

ciphertext. Data that is unintelligible to all except those who have the key to decode it to plaintext. Contrast with *plaintext*.

CL. See *control language*.

command processing program (CPP). A program that processes a command. This program performs some validity checking and processes the command so that the requested function is performed.

complement. A binary value that, in an exclusive-OR operation with a given binary value of the same length, produces a binary value of all ones.

control language (CL). The set of all commands with which a user requests system functions.

CPP. See *command processing program (CPP)*.

cross-domain key. A type of key-encrypting key used to encrypt a data-encrypting key that is being sent across a data line or being stored in a file.

cross-domain key table. A table in the system-supplied physical file QACRKTBL in library QUSRSYS used to store all key-encrypting keys other than the host master key and its variants. Each record of the file contains the name of the key, its use, and its value. There are three types of uses: sending, receiving, and personal identification numbers (PIN).

cryptanalyst. A specialist in solving cryptographic problems.

cryptographic algorithm. A set of rules that specify the mathematical steps required to encrypt and decrypt data. See also *algorithm*.

cryptography. The transformation of data to conceal its meaning; secret code.

Data Encryption Algorithm (DEA). Equivalent to the Data Encryption Standard. Adopted by the American National Standards Institute in 1981.

Data Encryption Standard (DES). A cryptographic algorithm designed to encrypt and decrypt data using a 64-bit key as specified in the Federal Information Processing Standard 46, January 15, 1977.

data-encrypting key. A key used to encrypt data that is not a cryptographic key. Used with the CPHDTA and GENMAC commands.

DEA. See *Data Encryption Algorithm (DEA)*.

decimalization table. A table of 16 decimal characters that is used to convert a hexadecimal value to a decimal value. Each hexadecimal digit is used as an offset into the (decimalization) table and is replaced with the value found there.

decrypt. To convert ciphertext into plaintext. Contrast with *encrypt*.

dedicated service tools (DST). The part of the service function used to service the system when the operating system is not working.

DES. See *Data Encryption Standard (DES).*

encrypt. To systematically scramble information so that it cannot be read without knowing the coding key.

exclusive-OR. A logic operator having the property that if P is a binary digit and Q is a binary digit and either P or Q is 1, but not both, then P exclusive-OR Q is 1.

hexadecimal. Pertaining to a numbering system with a base of 16.

host master key. A type of key-encrypting key used to encrypt data-encrypting keys and whose variants are used to encrypt all other key-encrypting keys stored on the system.

host master-key variant. A key-encrypting key derived from the host master key that is used to encrypt a certain type of cross-domain key.

ICV. See *initial chaining value.*

initial chaining value (ICV). An 8-byte, pseudo-random number used to start a cipher block chaining operation.

input PIN-protection key. A key-encrypting key that encrypts a personal identification number (PIN) that is received from another location. While a PIN is being used on the system, it remains encrypted under the input PIN-protection key.

interactive job. A job started for a person who signs on to a work station. Contrast with *batch job.*

key. A 64-bit value (containing 56 independent bits and 8 parity bits) used by the Data Encryption Algorithm to determine the output of the algorithm.

key schedule. Sixteen 8-byte keys created by the Data Encryption Algorithm from the supplied cryptographic key that are used to encrypt or decrypt the supplied data.

key table. See *cross-domain key table.*

key translation. The conversion of a data encrypting key from encryption under a previous key-encrypting key to encryption under another key-encrypting key.

key-encrypting key. A key used to encrypt another cryptographic key. See also *cross-domain key* and *host master key.*

licensed internal code. An instruction or group of instructions located in storage or device controllers that is shipped from the IBM Software Division with the operating system to control the operation of a device or controller. The licensed internal code cannot be called by the operating system or an application program. See also *model-unique licensed internal code.*

licensed program. An IBM-written program that performs functions related to processing user data.

MAC. See *message authentication code (MAC).*

machine interface (MI). The instruction set that tells the computer how to operate.

master key. See *host master key.*

message authentication code (MAC). The first 4 bytes from the last 8-byte block of ciphertext produced when encrypting a message using cipher block chaining, that is added to the end of the plaintext message from which it was created and used to detect whether the message was changed during transmission.

message authentication key. A data encrypting key used to encrypt data to produce a message authentication code.

MI. See *machine interface (MI).*

output PIN-protection key. A key encrypting key used to encrypt a PIN before it is sent to another location.

packed decimal format. Representation of a decimal value in which each byte within a field represents two numeric digits except the far right byte, which contains one digit in bits 0 through 3 and the sign in bits 4 through 7. For all other bytes, bits 0 through 3 represent one digit; bits 4 through 7 represent one digit. For example, the decimal value + 123 is represented as 0001 0010 0011 1111. Contrast with *zoned decimal format.*

pad. To fill unused positions in a field with dummy data, usually zeros or blanks.

personal identification number (PIN). A unique number assigned by an organization to an individual and used as proof of identity. PINs are commonly assigned by financial institutions to their customers.

PIN. See *personal identification number (PIN).*

PIN check length. The number of digits from the personal identification number that are verified.

PIN translation. The conversion of a personal identification number (PIN) encrypted under an input

PIN-protection key to encryption under an output PIN-protection key.

PIN-validation key. A key-encrypting key used to encrypt the validation data in the process of creating a customer's personal identification number (PIN).

plaintext. Data that can be read without decoding. Contrast with *ciphertext*.

pseudo-random number. A number that is obtained by some defined arithmetic process, but is effectively a random number for the purpose for which it is required. Contrast with *random number*.

random number. A number obtained by chance. Contrast with *pseudo-random number*.

receiving cross-domain key. A cross-domain key used to decrypt a data-encrypting key that was encrypted by another location.

seed. A value supplied on the GENCRSDMNK command to add a level of randomness to the creation of pseudo-random cross-domain keys.

sending cross-domain key. A cross-domain key used to encrypt a data-encrypting key before it is sent to another location.

session key. A data-encrypting key used to encrypt data before it is sent to another location.

System/36 environment. A function of the operating system that processes most of the System/36 operator control language (OCL) statements and procedure statements to run System/36 application programs and allows the user to process the control language (CL) commands. Contrast with *System/38 environment*.

System/38 environment. A function of the operating system that processes most of the System/38 control language (CL) statements and programs to run System/38 application programs. Contrast with *System/36 environment*.

validation data. Information about a customer used to create and verify the customer's personal identification number (PIN).

variant. See *host master-key variant*.

weak key. A value for a key-encrypting key that has known techniques that a code breaker can use to decrypt a data-encrypting key encrypted with this value.

zoned decimal format. A format for representing numbers in which the digit is contained in bits 4 through 7 and the sign is contained in bits 0 through 3 of the far right byte; bits 0 through 3 of all other bytes contain 1s (hex F). For example, in zoned decimal format, the decimal value of +123 is represented as 1111 0001 1111 0010 1111 0011. Same as unpacked decimal format. Contrast with *packed decimal format*.

Bibliography

This book is also referred to as *System/38 Environment Programming*.

| The books below are listed with their full title and base
| order number.

- *Publications General Information*, SC41-3000, lists all AS/400 publications.
- *CL Programming*, SC41-3721, provides a wide-range discussion of AS/400 programming topics, including a general discussion of objects and libraries, CL programming, working with objects in CL programs, and creating CL programs.
- *CL Reference*, SC41-3722, describes commands and parameters used for various OS/400 functions.
- *DB2/400 Database Programming*, SC41-3701, provides a detailed discussion of the AS/400 database organization, including information on how to create, describe, and update database files on the system.
- *Data Management*, SC41-3710, describes how to manage key aspects of the system. These include the following:
  - How to describe device files to the system
  - How to create jobs and output queues including information on spooling, job queues, and spooling output files
  - How to override and copy files
- *System Operation*, SC41-3203, provides information about how to use the system unit control panel, send and receive messages, respond to error messages, start and stop the system, and do system tasks.
- *System Startup and Problem Handling*, SC41-3206, provides information needed before the operating system is available, and information needed for solving problems when the operating system is not available.

Index

Reader Comments—We'd Like to Hear from You!

Application System/400
Cryptographic Support/400
Version 3

Publication No. SC41-3342-00

Overall, how would you rate this manual?

	Very Satisfied	Satisfied	Dissatis-fied	Very Dissatis-fied
Overall satisfaction				

How satisfied are you that the information in this manual is:

Accurate				
Complete				
Easy to find				
Easy to understand				
Well organized				
Applicable to your tasks				
T H A N K Y O U !				

Please tell us how we can improve this manual:

May we contact you to discuss your responses? __ Yes __ No

Phone: (____) ____________ Fax: (____) ____________

To return this form:

- Mail it
- Fax it
  United States and Canada: **800+937-3430**
  Other countries: **(+1)+507+253-5192**
- Hand it to your IBM representative.

Note that IBM may use or distribute the responses to this form without obligation.

Name

Address

Company or Organization

Phone No.

Reader Comments—We'd Like to Hear from You!
SC41-3342-00

Cut or Fold
Along Line

Fold and Tape **Please do not staple** Fold and Tape

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

ATTN DEPT 245
IBM CORPORATION
3605 HWY 52 N
ROCHESTER MN 55901-9986

Fold and Tape **Please do not staple** Fold and Tape

SC41-3342-00

Cut or Fold
Along Line

Program Number: 5763-CR1

Printed in U.S.A.

SC41-3342-00