Encryption for OpenVMS Installation and Reference Manual

Order Number: AA–EY97E–TE

July 2001

Revision/Update Information:	This manual supersedes the *Encryption for OpenVMS Installation and Reference Manual*, Version 1.3
Operating System:	OpenVMS VAX Versions 7.2, 7.3 OpenVMS Alpha Versions 7.2-1, 7.3
Software Version:	Encryption for OpenVMS Version 1.6

Compaq Computer Corporation
Houston, Texas

ZK6477

The Compaq *OpenVMS* documentation set is available on CD-ROM.

Contents

Preface vii

1 Product Overview

1.1 Encryption Process 1–1
1.1.1 DES Encryption Algorithm 1–1
1.1.2 Keys 1–1
1.1.3 Decryption 1–2
1.2 Authentication Process 1–3
1.3 Encryption Interfaces 1–3
1.4 Compatibility 1–3

2 Installing the Encryption for OpenVMS Software

2.1 Requirements 2–1
2.2 Before You Run the Installation Procedure 2–1
2.2.1 De-Installing Previous Versions 2–1
2.2.2 Pre-Installation: Installing onto OpenVMS VAX Systems 2–1
2.2.3 Pre-Installation: Loading the License PAK 2–2
2.2.3.1 Installing After You Load the License PAK 2–2
2.2.3.2 Installing Before You Load the License PAK 2–2
2.2.4 Disk onto Which You Install 2–3
2.3 Installing the Software 2–3
2.3.1 Installation Procedure 2–3
2.4 After You Run the Installation Procedure 2–5
2.4.1 Rereading the Release Notes 2–5
2.5 De-Installing 2–6

3 Using the Encryption for OpenVMS Features

3.1 Defining Keys 3–1
3.1.1 Specifying the Key Name 3–1
3.1.2 Specifying the Key Value 3–2
3.1.3 Verifying Key Creation 3–2
3.1.4 Specifying Key Storage Tables 3–3
3.1.5 Key Maintenance 3–4
3.2 Encrypting Files 3–4
3.2.1 Input File Specification 3–4
3.2.2 Output File Specification 3–4
3.2.3 Displaying Processing Information 3–5
3.2.3.1 FILES Keyword 3–5
3.2.3.2 STATISTICS Keyword 3–5

3.2.4 Specifying Files to Encrypt 3–6
3.2.4.1 /BACKUP Qualifier 3–6
3.2.4.2 /BEFORE Qualifier 3–6
3.2.4.3 /BY_OWNER Qualifier 3–7
3.2.4.4 /CONFIRM Qualifier 3–7
3.2.4.5 /EXCLUDE Qualifier 3–7
3.2.4.6 /EXPIRED Qualifier 3–8
3.2.4.7 /MODIFIED Qualifier 3–8
3.2.4.8 /SINCE Qualifier 3–9
3.2.5 Deleting Encrypted Files 3–9
3.2.5.1 /DELETE Qualifier 3–9
3.2.5.2 /ERASE Qualifier 3–10
3.2.6 Encryption Algorithms 3–10
3.2.7 Encryption Algorithm Qualifiers 3–11
3.2.7.1 /KEY_ALGORITHM Qualifier 3–12
3.2.7.2 /DATA_ALGORITHM Qualifier 3–12
3.2.8 File Compression 3–12
3.2.9 Displaying the Version Number 3–13
3.3 Authenticating Files 3–13
3.3.1 Associating MACs with Files 3–14
3.3.2 Checking Files 3–14
3.3.3 Specifying a File for MACs Generated from File Contents 3–15
3.3.4 Specifying a Security MAC File 3–15
3.3.5 Specifying a Listing File 3–15
3.3.6 Logging the Authentication Operation 3–16
3.4 Deleting Key Definitions 3–16
3.5 Decrypting Files 3–17
3.5.1 Input File Specification 3–17
3.5.2 Output File Specification 3–18
3.5.3 Displaying Processing Information 3–18
3.5.3.1 FILES Keyword 3–18
3.5.3.2 STATISTICS Keyword 3–18
3.5.4 Specifying Files to Decrypt 3–19
3.5.4.1 /BACKUP Qualifier 3–19
3.5.4.2 /BEFORE Qualifier 3–20
3.5.4.3 /BY_OWNER Qualifier 3–20
3.5.4.4 /CONFIRM Qualifier 3–20
3.5.4.5 /EXCLUDE Qualifier 3–21
3.5.4.6 /EXPIRED Qualifier 3–21
3.5.4.7 /MODIFIED Qualifier 3–22
3.5.4.8 /SINCE Qualifier 3–22
3.5.5 Deleting Decrypted Files 3–22
3.5.5.1 /DELETE Qualifier 3–23
3.5.5.2 /ERASE Qualifier 3–23
3.5.6 Algorithm Qualifiers 3–23
3.6 Encrypting Save Sets 3–24
3.6.1 Restoring Files 3–25
3.6.2 Encrypting Distribution Files 3–26

4 Programming with Encryption for OpenVMS Routines

4.1 How the Routines Work 4–2
4.1.1 DES Key and Data Semantics 4–2
4.2 Maintaining Keys 4–3
4.3 Operations on Files 4–4
4.4 Operations on Records and Blocks 4–4
4.5 Routine Descriptions 4–5
4.5.1 Specifying Arguments 4–5
4.5.2 Bitmasks 4–5
4.5.3 Error Handling 4–5
ENCRYPT$DECRYPT 4–6
ENCRYPT$DECRYPT_ONE_RECORD 4–8
ENCRYPT$DEFINE_KEY 4–10
ENCRYPT$DELETE_KEY 4–12
ENCRYPT$ENCRYPT 4–13
ENCRYPT$ENCRYPT_FILE 4–15
ENCRYPT$ENCRYPT_ONE_RECORD 4–18
ENCRYPT$FINI 4–20
ENCRYPT$GENERATE_KEY 4–21
ENCRYPT$INIT 4–23
ENCRYPT$STATISTICS 4–26

A Command Reference

BACKUP /ENCRYPT A–2
DECRYPT A–3
ENCRYPT A–6
ENCRYPT /AUTHENTICATE A–9
ENCRYPT /CREATE_KEY A–12
ENCRYPT /REMOVE_KEY A–14

B Error Messages

B.1 ENCRYPT and DECRYPT Messages B–1
B.2 BACKUP Utility Messages B–10

Index

Figures

1–1 Encrypting a File 1–2
1–2 Decrypting a File 1–3
3–1 Relationship of Keys and Algorithms 3–11
4–1 OpenVMS Numbering Overlay on FIPS-46 Numbering 4–3
4–2 NBS Numbering Overlay on an OpenVMS Quadword 4–3

Tables

1–1 Components of the Encryption Operation 1–2
4–1 Comparison of NBS and OpenVMS Numbering Conventions 4–2
4–2 Routines for Maintaining Keys 4–3
4–3 ENCRYPT$DEFINE_KEY Flags 4–11
4–4 ENCRYPT$ENCRYPT_FILE Flags 4–16

Preface

The Encryption for OpenVMS product (Encryption) is a standalone layered product that runs on OpenVMS Alpha and OpenVMS VAX systems.

Purpose

The purpose of this manual is to explain the Encryption for OpenVMS product, show you how to use it, and describe how to write programs with its application programming interface.

Intended Audience

This document is for OpenVMS programmers, system managers, and users of this security software.

Document Structure

This manual consists of four chapters and two appendixes.

- Chapter 1 introduces the features of the Encryption for OpenVMS product.
- Chapter 2 shows how to install the software.
- Chapter 3 describes how to use the Encryption features.
- Chapter 4 describes how to develop programs with the Encryption for OpenVMS programming interface and provides complete reference information for the routines.
- Appendix A describes the syntax of the Encryption commands.
- Appendix B explains the error messages generated by the Encryption commands.

Related Information

For additional information about the Encryption for OpenVMS product refer to the Software Product Description (SPD) 26.74.xx.

For additional information about Compaq *OpenVMS* products and services, access the Compaq website at the following location:

`http://www.openvms.compaq.com/`

Terminology

This document uses the terms:

- **Encryption** — the software in the Encryption for OpenVMS product. This software is a separately orderable OpenVMS layered product.
- **Command** — a Digital Command Language (DCL) command. Any other command is identified as such.

Reader's Comments

Compaq welcomes your comments on this manual. Please send comments to either of the following addresses:

Internet	**openvmsdoc@compaq.com**
Mail	Compaq Computer Corporation OSSG Documentation Group, ZKO3-4/U08 110 Spit Brook Rd. Nashua, NH 03062-2698

How To Order Additional Documentation

Visit the following World Wide Web address for information about how to order additional documentation:

`http://www.openvms.compaq.com/`

If you need help deciding which documentation best meets your needs, call 800-282-6672.

Conventions

The following conventions are used in this manual:

Ctrl/*x*	A sequence such as Ctrl/*x* indicates that you must hold down the key labeled Ctrl while you press another key or a pointing device button.
PF1 *x* or GOLD *x*	A sequence such as PF1 *x* or GOLD *x* indicates that you must first press and release the key labeled PF1 or GOLD and then press and release another key or a pointing device button. GOLD key sequences can also have a slash (/), dash (–), or underscore (_) as a delimiter in EVE commands.
$\boxed{\text{Return}}$	In examples, a key name enclosed in a box indicates that you press a key on the keyboard. (In text, a key name is not enclosed in a box.)
. . .	Horizontal ellipsis points in examples indicate one of the following possibilities: • Additional optional arguments in a statement have been omitted. • The preceding item or items can be repeated one or more times. • Additional parameters, values, or other information can be entered.

. . .	Vertical ellipsis points indicate the omission of items from a code example or command format; the items are omitted because they are not important to the topic being discussed.
( )	In command format descriptions, parentheses indicate that, if you choose more than one option, you must enclose the choices in parentheses.
[ ]	In command format descriptions, brackets indicate optional elements. You can choose one, none, or all of the options. (Brackets are not optional, however, in the syntax of a directory name in an OpenVMS file specification or in the syntax of a substring specification in an assignment statement.)
{ }	In command format descriptions, braces indicate a required choice of options; you must choose one of the options listed.
boldface text	Boldface text represents the introduction of a new term or the name of an argument, an attribute, or a reason.
italic text	Italic text indicates important information, complete titles of manuals, or variables. Variables include information that varies in system output (Internal error *number*), in command lines (/PRODUCER=*name*), and in command parameters in text (where *device-name* contains up to five alphanumeric characters).
UPPERCASE TEXT	Uppercase text indicates a command, the name of a routine, the name of a file, the name of a node, the name of a user account, or the abbreviation for a system privilege.
`Monospace type`	Monospace type indicates command examples and interactive screen displays. In the C programming language, monospace type in text identifies the following elements: keywords, the names of independently compiled external functions and files, syntax summaries, and references to variables or identifiers introduced in an example.
-	A hyphen at the end of a command format description, command line, or code line indicates that the command or statement continues on the following line.
numbers	All numbers in text are assumed to be decimal unless otherwise noted. Nondecimal radixes — binary, octal, or hexadecimal — are explicitly indicated.

1 Product Overview

The OpenVMS operating system provides several data protection schemes. For example, by using UIC-based protection you can protect data by controlling access to files. You can use ACLs to refine access control to specific groups or individual users. For a protection scheme with yet greater security for your data, you can encrypt the files. Encrypting a file transforms it into unrecognizable, unintelligible data, even if someone manages to gain access to it.

1.1 Encryption Process

The process of encryption takes readable data, called **plaintext**, and uses a mathematical algorithm to transform the plaintext into an unreadable, unintelligible form, called **ciphertext**.

To encrypt the plaintext data, the encryption operation requires a **key**. The key is a variable that controls the encryption operation. The same plaintext, encrypted with different keys, results in different ciphertext. In addition, repeated encryption of the same plaintext with the same key also results in different ciphertext each time.

1.1.1 DES Encryption Algorithm

The algorithm used by the Encryption for OpenVMS product is a software implementation of the Data Encryption Standard (DES) defined by the National Bureau of Standards (NBS). The NBS document FIPS-PUB-46 describes the operation of the DES algorithm in detail.

Because the DES algorithm is public knowledge, the security of your ciphertext files depends on the keys you define.

1.1.2 Keys

The Encryption for OpenVMS implementation uses two keys:

- Key that you provide
- Key that the software randomly generates, called the **data key**.

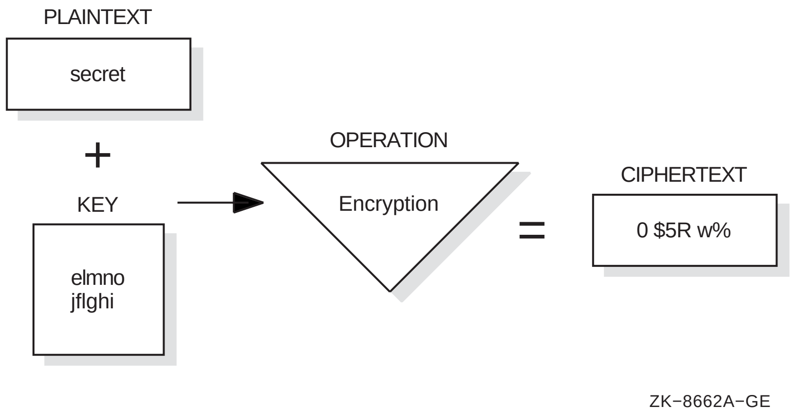
The key you provide encrypts the data key, which is stored in the first block of the ciphertext file. The process uses the encrypted data key to encrypt the file. You have the option to encrypt either the data key or the file. Table 1–1 shows the components of the encryption process.

Table 1–1 Components of the Encryption Operation

Input	Algorithm	Output
User-supplied data key	Key encryption	Encrypted key
Data (plaintext) and the encrypted data key	Data encryption	Encrypted file

Figure 1–1 illustrates the data encryption operation. In this example, the input file contains the text "secret" and the key has been defined as "elmno jflghi." The output file is unreadable text.

Figure 1–1 Encrypting a File



1.1.3 Decryption

To gain access to the data in an encrypted file, reverse the encryption process by performing the decryption process. Decryption uses a mathematical encryption algorithm to change ciphertext into the original plaintext.

Before decrypting a file, the software checks the validity of the key you provide. This validation is a checksum operation on the encrypted data stored in the first block of the ciphertext file.

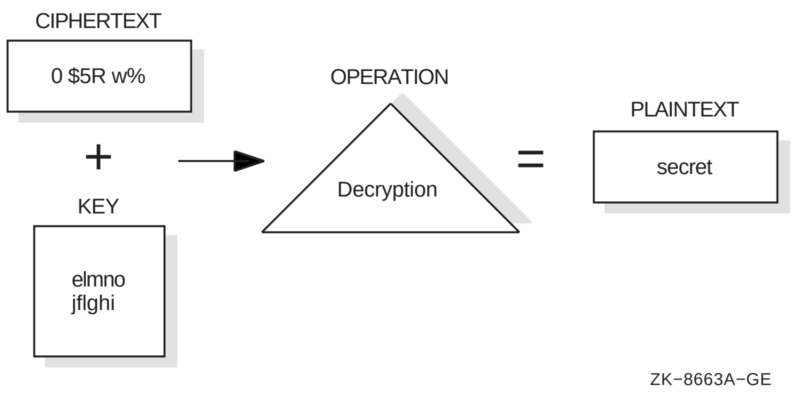
When you specify the DES algorithm to decrypt a file, use the key that is identical to the one used in the original encryption process.

Warning

Only the correct key can decrypt your file. If you lose or forget the key, you cannot gain access to the data in any understandable, useful form.

Figure 1–2 shows the data decryption operation. In this example, the input file holds unreadable text. The key, "elmno jflghi," is the same key that was used to encrypt this file. The output file contains the readable text "secret."

Figure 1–2 Decrypting a File



1.2 Authentication Process

The Encryption for OpenVMS software detects any modification made to both plaintext and ciphertext files. This process is called **authentication**. Authentication checks for and reports on any changes to:

- File data
- File location
- Authentication key
- Security settings

The software calculates two Message Authentication Codes (MACs): one based on file contents and one based on security settings. The software then associates them with one or more files and stores this information. When you subsequently check file integrity, the software recalculates the MACs and compares them against the stored codes.

For information about how to authenticate files, see Section 3.3.

1.3 Encryption Interfaces

To define and delete keys and to encrypt and decrypt files, use the following Encryption interfaces:

- DCL commands — for interactive encryption functions. These commands encrypt files and backup save sets (see Chapter 3).
- Callable routines — for application programming. These routines encrypt files and small blocks (Chapter 4).

1.4 Compatibility

The Encryption for OpenVMS software includes full backward compatibility with previous releases of the product.

In addition, encrypted files are fully compatible between OpenVMS systems. You can copy them from system to system and do all remote file operations that OpenVMS systems support for other kinds of files. In addition, you can encrypt files on one system and decrypt them on another system that also runs the Encryption software.

Inter-system encryption operations with non-OpenVMS platforms are not supported.

2 Installing the Encryption for OpenVMS Software

The Encryption for OpenVMS layered product uses the POLYCENTER Software Installation procedure.

For CD kits, the POLYCENTER Software Installation procedure automatically identifies and installs the distribution kit that is appropriate to your platform — either an OpenVMS Alpha or OpenVMS VAX system.

2.1 Requirements

Before you start the procedure, ensure that you meet the following installation requirements:

- Operating system — one of the following:
  - OpenVMS Alpha Version 7.2-1, 7.3
  - OpenVMS VAX Version 7.2, 7.3
- License (see Software Product Description [SPD] 26.74.xx)
- Disk space
  - 2000 blocks
- If the Encryption for OpenVMS software is not currently installed on your system, the installation procedure may require an additional 1000 blocks.

For complete information about hardware and software requirements, see the SPD.

2.2 Before You Run the Installation Procedure

Before you start the installation, complete the following pre-installation tasks.

2.2.1 De-Installing Previous Versions

Compaq recommends that you de-install previous versions of Encryption for OpenVMS before you install Version 1.6. See Section 2.5 for de-installation information.

2.2.2 Pre-Installation: Installing onto OpenVMS VAX Systems

If you are installing the kit onto an OpenVMS VAX system, follow these steps:

1. Check that the operating system is registered with the POLYCENTER Software. Enter:

   ```
   $ PRODUCT SHOW PRODUCT VMS
   ```

2. If the output shows that no products were found, register the operating system:

   ```
   $ PRODUCT REGISTER PRODUCT VMS /SOURCE=SYS$UPDATE:
   ```

2.2.3 Pre-Installation: Loading the License PAK

Before you install onto a newly licensed node or cluster, register the Product Authorization Key (License PAK). Use the License Management facility (LMF). If you are installing Encryption for OpenVMS Version 1.6 as an update onto a node or cluster already licensed for this software, you have already completed the License PAK registration requirements.

Your PAK might be shipped along with the kit if you ordered the license and media together. Otherwise, it is shipped separately to a location based on your license order.

To register a license, follow these steps:

1. Log in to the SYSTEM account.
2. Do one of the following steps:
   - Run SYS$UPDATE:VMSLICENSE.COM and enter the data from your License PAK.
   - Enter the DCL LICENSE REGISTER command.
   - To run the software on multiple cluster nodes, perform a LICENSE LOAD on each node.

For complete information about LMF, see the *OpenVMS License Management Utility Manual*.

The POLYCENTER Software Installation procedure automatically calls the Installation Verification Procedure (IVP). The availability of a valid license is checked. If your Encryption license is registered, the procedure runs the IVP. If your license is not registered, you receive a message showing that the installation is complete, but verification cannot be performed.

2.2.3.1 Installing After You Load the License PAK

When you register your PAK before starting the installation, the IVP might display these messages during the installation procedure:

```
% Verification of installation starting
```

Displayed when IVP is run.

```
% Successful verification of installation
```

Displayed when the IVP is run without any errors.

2.2.3.2 Installing Before You Load the License PAK

If you install the Encryption software before registering your PAK, the IVP might display these messages during the installation procedure:

```
% Installation complete, but verification cannot be run because
  license for ENCRYPTION is not available
```

A valid license is not loaded.

```
% Installation complete, but verification cannot be run because
  test data files are not in SYS$COMMON:[SYSTEST.ENCRYPTION]
```

You set the destination for the installed files to a device other than SYS$COMMON.

```
% Verification of installation failed
```

The IVP cannot successfully complete.

2.2.4 Disk onto Which You Install

Compaq strongly suggests that you install the software onto the system disk SYS$COMMON:. If you use a different device, neither the IVP nor the Encryption software can run until you complete these additional steps:

- Edit the IVP.
- Edit the Encryption startup procedure.
- Edit your system startup procedure.
- Modify other associated files and utilities.

2.3 Installing the Software

The Encryption for OpenVMS CD kits support both OpenVMS Alpha systems and OpenVMS VAX systems (see Section 2.3.1).

2.3.1 Installation Procedure

To start the procedure, follow these steps:

1. Log in to the SYSTEM account.
2. Refer to the Software Product Library Master Index for the correct CD number and directory name for the Encryption software. Insert the CD into the drive.
3. Mount the CD using the volume label of the CD containing the encryption software. Instructions on how to mount the CD are contained in the Software Product Library Getting Started document. Additional information on mounting the discs is contained in the [README] directory on disc 1. The following example assumes that SPRING is your system's name and DKA400 is the name of the device onto which you are mounting the compact disc medium.

   ```
   $ MOUNT /SYSTEM DKA400: volume-label

   %MOUNT-I-WRITELOCK, volume is write locked
   %MOUNT-I-MOUNTED, volume-label mounted on _SPRING$DKA400:
   ```

4. Check the directory of the CD. Enter the DIRECTORY command and specify the installation device, for example:

   ```
   $ DIRECTORY DKA400:[ENCRYPT0106...]

   Directory DKA400:[ENCRYPT0106]

   DOCUMENTATION.DIR;1 KIT.DIR;1

   Total of 2 files.

   Directory DKA400:[ENCRYPT0106.DOCUMENTATION]
   ```

```
CPQ-VMS-ENCRYPT-V0106_RELEASE_NOTES.TXT;1
CPQ-VMS-ENCRYPT-V0106_RELEASE_NOTES.PS;1
CPQ-VMS-ENCRYPT-V0106_SPD.PS;1
CPQ-VMS-ENCRYPT-V0106_SPD.TXT;1
CPQ-VMS-ENCRYPT-V0106_COVER_LETTER.TXT;1
CPQ-VMS-ENCRYPT-V0106_COVER_LETTER.PS;1

Total of 6 files.

Directory DKA400:[ENCRYPT0106.KIT]

CPQ-AXPVMS-ENCRYPT-V0106--1.PCSI;1
CPQ-VAXVMS-ENCRYPT-V0106--1.PCSI;1

Total of 2 files.

Grand total of 3 directories, 10 files.
```

5. Finish reading the Release Notes, if you have not already done so.

   To read them on-line, enter the TYPE command, as follows:

```
$ TYPE /PAGE -
_$ DKA400:[ENCRYPT0106.DOCUMENTATION]CPQ-VMS-ENCRYPT-V0106.RELEASE_NOTES
```

   To print a PostScript file, enter the PRINT command. The following example prints the Release Notes, mounted on DKA400:, to print queue PRINTER_7:

```
$ PRINT /QUEUE=PRINTER_7 -
_$ DKA400:[ENCRYPT0106.DOCUMENTATION]DEC-VMS-ENCRYPT-V0106_RELEASE_NOTES.PS
```

6. Run the installation procedure. Use the same command line whether you are installing onto an OpenVMS Alpha system or an OpenVMS VAX system. The POLYCENTER Software Installation procedure selects the appropriate save set for your platform. For example, if the CD is mounted on device DKA400:, enter:

```
$ PRODUCT INSTALL ENCRYPT /SOURCE=DKA400:ENCRYPT0106
```

   If you omit the /SOURCE qualifier, the POLYCENTER Software Installation procedure searches in the location defined by the logical name PCSI$SOURCE. If you omit both /SOURCE and PCSI$SOURCE, the procedure searches your current default directory for the Encryption for OpenVMS kit.

   In the following example, the command installs the software from the CD-ROM with device name DKA400: onto an OpenVMS Alpha system.

```
$ PRODUCT INSTALL ENCRYPT /SOURCE=DKA400:[ENCRYPT0106.KIT]

The following product has been selected:

    CPQ AXPVMS ENCRYPT V1.6                 Layered Product

Do you want to continue? [YES]) Return

Configuration phase starting ...

You will be asked to choose options, if any, for each selected product and for
any products that may be installed to satisfy software dependency requirements.

CPQ AXPVMS ENCRYPT V1.6: Compaq Encryption for OpenVMS Alpha

    (c) Compaq Computer Corporation 2001.  All rights reserved.

 Do you want the defaults for all options? [YES]) Return

 Do you want to review the options? [NO] Return
```

```
Execution phase starting ...

The following product will be installed to destination:
    CPQ AXPVMS ENCRYPT V1.6               DISK$ALPHA:[VMS$COMMON.]

Portion done: 0%...10%...20%...30%...40%...50%...60%...70%...80%...90%...100%

The following product has been installed:
   CPQ AXPVMS ENCRYPT V1.6                Layered Product

CPQ AXPVMS ENCRYPT V1.6: Compaq Encryption for OpenVMS Alpha

   @ SYS$STARTUP:ENCRYPT_START.COM required in system startup

   Note: Upgrading OpenVMS requires Encryption product reinstallation

   Refer to SYS$HELP: CPQ-VMS-ENCYRPT-V0106.RELEASE.NOTES for more information.
```

7. To see a list of the files installed, issue the following command:

```
PRODUCT LIST ENCRYPT/SOURCE=[pcsi kit location]
```

2.4 After You Run the Installation Procedure

When the installation is finished, before you run the *Encryption for OpenVMS* software, follow these post-installation steps:

1. Edit the system startup procedure, SYS$STARTUP:SYSTARTUP_VMS.COM.
   - Add the following line:

     $ @SYS$SYSROOT:[SYS$STARTUP]ENCRYPT_START.COM

     As part of installation, PCSI runs the Encryption startup procedure. However, ENCRYPT_START.COM also needs to be run upon each reboot. Adding this instruction to your system startup procedure ensures that ENCRYPT_START.COM runs whenever your system reboots.
   - If you plan to use the Encryption programming interface from an image installed with privileges, add this line:

     $ INSTALL ADD SYS$LIBRARY:ENCRYPSHR.EXE /OPEN /HEADER /SHARED

     The programming interface resides in the ENCRYPSHR shareable image. The INSTALL ADD command installs the ENCRYPSHR shareable image as a known image.
2. Log out and then log in again.

2.4.1 Rereading the Release Notes

After installation, you can access the *Encryption for OpenVMS Release Notes*, even if you dismounted the distribution medium, or if you deleted a copy of the Release Notes from the Encryption directory. Use the PRODUCT EXTRACT RELEASE_NOTES command to copy the document from the installed kit to a file.

The default file name is DEFAULT.PCSI$RELEASE_NOTES, in your current directory. To specify a different name for the file, use the /FILE qualifier. Follow these steps:

1. Ensure that one of the following is true:
   - The logical name PCSI$SOURCE is defined to point to the directory containing the .PCSI files.
   - The directory containing the .PCSI files is your default directory.

2. Enter the PRODUCT EXTRACT RELEASE_NOTES command. For example:

```
$ PRODUCT EXTRACT RELEASE_NOTES CPQ-VMS-ENCRYPT-V0106.RELEASE_NOTES -
_$ /FILE=[ENCRYPT0106]RELEASE_NOTES.TXT
```

2.5 De-Installing

To de-install Encryption files, use the PRODUCT REMOVE command:

```
$ PRODUCT REMOVE ENCRYPT

The following product has been selected:
    CPQ AXPVMS ENCRYPT V1.6                Layered Product

Do you want to continue? [YES] Return

The following product will be removed from destination:
    CPQ AXPVMS ENCRYPT V1.6                DISK$COE_X92N:[VMS$COMMON.]

Portion done: 0%...20%...30%...40%...50%...60%...70%...80%...90%...100%

The following product has been removed:
    CPQ AXPVMS ENCRYPT V1.6                Layered Product
```

3 Using the Encryption for OpenVMS Features

Interactively using the features of the Encryption for OpenVMS product consists of the following tasks:

- Defining keys (Section 3.1)
- Encrypting files (Section 3.2)
- Authenticating files (Section 3.3)
- Deleting key definitions (Section 3.4)
- Decrypting files (Section 3.5)
- Encrypting save sets (Section 3.6)

Use DCL commands to accomplish these tasks. For information about adding encryption and decryption processes to your software applications, see Chapter 4.

3.1 Defining Keys

To define a key, enter the ENCRYPT /CREATE_KEY command:

ENCRYPT /CREATE *key-name key-value* [*qualifiers*]

where

key-name	is the name of the key.
key-value	is the value you assign to the key.
qualifiers	are options that control the format of the key value or where the key is stored.

3.1.1 Specifying the Key Name

To specify *key-name* on the ENCRYPT /CREATE_KEY command line, specify a character string using the following rules:

- Valid length: 1 to 243 characters.
- Valid: alphanumeric characters, dollar signs, and underscores.
- Case sensitive: no.

To help you remember the name, use one that has meaning to you.

Note

Key names beginning with ENCRYPT$ are reserved for Compaq.

3.1.2 Specifying the Key Value

To specify a *key-value* on the ENCRYPT /CREATE_KEY command line, use either a text string or a hexadecimal constant, using the following rules:

ASCII text string (default):

- Length: 8 to 240 characters.
- The string is not case sensitive.
- If you use any non-alphanumeric characters, for example, space characters, enclose them in quotation marks (" ").

Example: This command defines a key named HAMLET with character string value `And you yourself shall keep the key of it`:

```
$ ENCRYPT /CREATE_KEY HAMLET

_ Key value: "And you yourself shall keep the key of it"
```

Hexadecimal constant

- Use the /HEXADECIMAL qualifier.
- Valid characters: 0 to 9, A to F.
- Valid minimum length: 15 characters.
- Do **not** enclose the value in quotation marks.

Example: The following command defines a key named ARCANE with hexadecimal value 2F4A98F46BBC11D:

```
$ ENCRYPT /CREATE_KEY /HEX ARCANE 2F4A98F46BBC11D
```

In addition, when you specify *key-value*, do not use **weak keys**. These are key values with a pattern of repeated characters or groups of characters. Using a pattern results in an encrypted form that might be easy for unauthorized users to decrypt. For example, the hexadecimal constant 0101010101010101 and the text string 'abcabcabc' are weak keys.

Using weak keys might produce the following consequences:

- Security of encrypted data may be at risk.
- Encryption may be the same as decryption.
- Encryption with one weak key followed by encryption with another weak key may result in the original plaintext.

Compaq supplies a table of known weak keys with the Encryption software. The software checks keys you define against this table and displays an error message when you supply a weak key.

3.1.3 Verifying Key Creation

To verify the successful creation of a key, use the /LOG qualifier. For example, this command reports that the key HAMLET is defined:

```
$ ENCRYPT /CREATE_KEY /LOG HAMLET

_ Key value: "And you yourself shall keep the key of it"
%ENCRYPT-S-KEYDEF, key defined for key name = HAMLET
```

3.1.4 Specifying Key Storage Tables

When you define a key, it is stored in encrypted form in a key storage table. The key value is stored under the key name. When you encrypt files, the process takes this stored information and does the following:

- It compresses the key value taken from the key storage table into a key consisting of 8 bytes of binary digits.
- It ensures the odd parity of each byte by modifying one of two things for each byte:
  - Sign bit, as needed (default)
  - Low bit (bit 0) (if you specify the /HEXADECIMAL qualifier)
- For text string key values, it converts letters to uppercase, reduces multiple consecutive spaces to one space, removes some punctuation characters, and compresses the key string.

  As a result, you do not need to remember the exact syntax of the key value. For example, if you define a key value with two spaces between each word, remembering this spacing is not required to respecify the key.

Key storage tables determine which users can access keys. The following key storage tables control user access:

- Process key storage table (default) — accessible only to the process that defined the keys within the table.

  If you are defining a key that is intended for use by other processes, specify the appropriate qualifier (/JOB, /GROUP, or /SYSTEM) so that the intended users of the key can access it.
- Job key storage table — accessible only to processes within the same job tree as the process that defined the keys within the table.
- Group key storage table — accessible to users in the same UIC group as the process that defined the keys in the table.
- System storage table — accessible to all system users.

To enter keys into the key storage tables, use the following ENCRYPT /CREATE_KEY qualifiers:

- /PROCESS
- /JOB
- /GROUP (requires GRPNAM or SYSPRV privilege)
- /SYSTEM (requires SYSPRV privilege)

  Defines a key that anyone working on the system can use to encrypt his or her files. Because the key is stored in encrypted form, they cannot see the value of the key. The key is available for use until the system is rebooted.

  For example, the following command defines a key named SYSMASTER and places it in the system key storage table.

  ```
  $ ENCRYPT /CREATE_KEY /SYSTEM SYSMASTER

  _$ Key Value: "The human heart has hidden treasures, in secret kept,
  in silence sealed"
  ```

3.1.5 Key Maintenance

When you encrypt a file, the key you use is like a password to that file. It is important to keep it secret. In addition, ensure that you remember the key value. You need both the key and the value to decrypt the file.

A key stored in the process key storage table lasts for the life span of the process that defined the keys in the table. Like other process-specific structures, the process key storage table disappears when you log out.

Key values that are meaningful to you are the most memorable, but avoid easily guessed choices such as your nickname or the make of your car. Never post a key name or value in your office or store it online. Like operating system passwords, increasing the length of a key value lessens the possibility of discovery.

The DES algorithm requires that a key value have a minimum length of eight non-null characters. To improve the security of the key value, specify more than eight.

3.2 Encrypting Files

After you define a key with the ENCRYPT /CREATE_KEY command, use this key to encrypt files. Enter the ENCRYPT command. In addition to the key, specify a plaintext file. The syntax of the ENCRYPT command is as follows:

ENCRYPT *file-spec key-name* [*qualifiers*]

where

file-spec	is the plaintext input file specification.
key-name	is the name of the key.
qualifiers	are options that control the encryption process or the selection of files you want to encrypt

3.2.1 Input File Specification

For the plaintext file specified on the ENCRYPT command line, use a file that resides on disk and that is not a directory file.

To specify multiple input files, use wildcard characters in the file specification. To control file selection, specify the appropriate ENCRYPT command qualifiers (see Section 3.2.4). Do not use wildcard characters to specify directory files or files containing bad blocks.

3.2.2 Output File Specification

The result of the encryption operation is a ciphertext file. One ciphertext file is created for each input file that is encrypted.

By default, the ENCRYPT command writes each ciphertext file to a separate output file with the same name except that it has a version number one higher than that of the current input file.

To specify an alternate output file specification, use the /OUTPUT qualifier. Specify only the file specification parts that you want to change from the defaults. For example, the following command encrypts all the files in the current directory that match the wildcard file specification *.COM. The /OUTPUT qualifier specifies that any output files created have a file type of .ENC. FRANCISSCOTT is the key used to encrypt the files.

```
$ ENCRYPT *.COM /OUTPUT=.ENC FRANCISSCOTT
```

Do not specify a file that already exists. For example, you cannot name the output file NEWS.DAT;2 if NEWS.DAT;2 already exists. However, specifying NEWS.DAT as both the input and output files is valid.

3.2.3 Displaying Processing Information

By default, information about the encryption operation is not displayed. To display information about file encryption operations on SYS$COMMAND, use the /SHOW qualifier. The /SHOW qualifier has the format:

/SHOW=*keyword*

or

/SHOW=(*keyword-list*)

Specify one or more of the following keywords:

- FILES (Section 3.2.3.1)
- STATISTICS (Section 3.2.3.2)

3.2.3.1 FILES Keyword

The FILES keyword displays the file specifications of the input and output files. For example, /SHOW=FILES in the following command specifies that each input and output file specification be displayed as it is encrypted.

```
$ ENCRYPT /SHOW=FILES *.COM FRANCISSCOTT

%ENCRYPT-S-ENCRYPTED, DISK2:[FLYNN]MOVE.COM.2 encrypted to DISK2:[FLYNN]MOVE.COM;3 (8 blocks)
.
.
.
```

3.2.3.2 STATISTICS Keyword

Use the STATISTICS keyword to display encryption stream statistics after the completion of each file operation. The statistics displayed are:

- Bytes processed
- Internal records processed
- CPU time consumed within the encryption algorithm

The following command specifies that encryption stream statistics be displayed on SYS$COMMAND.

```
$ ENCRYPT /SHOW=STATISTICS *.COM FRANCISSCOTT

%ENCRYPT-S-STATISTICS, encryption stream statistics:
        Total Records: 65
        Total Bytes: 4083
        Total Time: 00:00:01.63
.
.
.
```

3.2.4 Specifying Files to Encrypt

To specify multiple input files, use the ENCRYPT command with wildcard characters in the input file specification.

The following ENCRYPT command qualifiers can help you select files:

- /BACKUP (Section 3.2.4.1)
- /BEFORE (Section 3.2.4.2)
- /BY_OWNER (Section 3.2.4.3)
- /CONFIRM (Section 3.2.4.4)
- /EXCLUDE (Section 3.2.4.5)
- /EXPIRED (Section 3.2.4.6)
- /MODIFIED (Section 3.2.4.7)
- /SINCE (Section 3.2.4.8)

3.2.4.1 /BACKUP Qualifier

The /BACKUP qualifier selects files for encryption according to the date of their most recent backup. This qualifier is meaningful only when used with either the /BEFORE or the /SINCE qualifier. The /BACKUP qualifier has the format:

/BACKUP /BEFORE[=*time*]

or

/BACKUP /SINCE[=*time*]

where

time is an OpenVMS time.

For more information on time specifications, see the *OpenVMS DCL Dictionary*. If you do not specify a time, TODAY is used. TODAY is the current day, month, and year at 00:00:00.

The following command selects for encryption all files in the current directory matching the wildcard file specification of *.COM that had backup copies made before 00:00:00 15-APR-2001.

```
$ ENCRYPT /BACKUP /BEFORE=15-APR-2001 *.COM FRANCISSCOTT
```

Do not use the /BACKUP qualifier with either the /EXPIRED or the /MODIFIED qualifier.

3.2.4.2 /BEFORE Qualifier

The /BEFORE qualifier selects files for encryption that have a creation time before the time specified with the qualifier. The /BEFORE qualifier has the format:

/BEFORE[=*time*]

where

time is an OpenVMS time.

For more information on time specifications, see the *OpenVMS DCL Dictionary*. If you do not specify a time, TODAY is used. TODAY is the current day, month, and year at 00:00:00.

The following command selects for encryption all files in the current directory matching the wildcard file specification of *.COM that were created before 00:00:00 15-APR-2001.

```
$ ENCRYPT /BEFORE=15-APR-2001 *.COM FRANCISSCOTT
```

3.2.4.3 /BY_OWNER Qualifier

The /BY_OWNER qualifier allows you to select files for encryption that have a particular owner User Identification Code (UIC). If no UIC is specified with the qualifier, the UIC of the current process is used. The /BY_OWNER qualifier has the format:

/BY_OWNER=*uic*

where *uic* is the UIC of the owner of the file.

For more information on specifying UIC format, see the *OpenVMS DCL Dictionary*.

The following command selects for encryption all files in the current directory owned by the user whose UIC is [FLYNN] that match the wildcard file specification of *.COM.

```
$ ENCRYPT /BY_OWNER=[FLYNN] *.COM FRANCISSCOTT
```

3.2.4.4 /CONFIRM Qualifier

By default, all input files specified on the command line are processed without confirming that those files are selected for encryption. Use the /CONFIRM qualifier if you want a prompt with the name of each file selected for encryption. Your response determines whether or not a particular file is encrypted, as follows:

Response	Meaning
YES	Encrypt the file.
NO or Return	Do not encrypt the file. This is the default.
QUIT or Ctrl/Z	Do not encrypt the file or any subsequent files.
ALL	Encrypt the file and all subsequent files.

The following command selects for encryption all files in the current directory matching the wildcard file specification of *.COM. Because the /CONFIRM qualifier is specified, the user is prompted on a file-by-file basis to confirm that each file is to be encrypted. Because the prompt is answered in the affirmative for the file MOVE.COM;3, the output file MOVE.COM;4 is created.

```
$ ENCRYPT /CONFIRM *.COM FRANCISSCOTT
Encrypt DISK2:[FLYNN]MOVE.COM;3 ? [N] YES
```

3.2.4.5 /EXCLUDE Qualifier

Use the /EXCLUDE qualifier to exclude one or more files from an encryption operation. If a file matches the file specification provided with the /EXCLUDE qualifier, the file will not be encrypted. The /EXCLUDE qualifier has the format:

/EXCLUDE=(*file-spec*[,...])

where

file-spec is the name of the file to remain unencrypted.

Wildcard characters are allowed in the file specification. There is no default for the file specification. Because directory files are never encrypted, you need not specify them with the /EXCLUDE qualifier. However, if you do specify `/EXCLUDE=*.DIR`, you will not get the warning message `%ENCRYPT-W-FILNODIR, file encryption of directories is not supported, filename.dir`.

The following command selects for encryption all files in the current directory that match the wildcard file specification of *.COM, except LOGIN.COM, which is specified with /EXCLUDE.

```
$ ENCRYPT /EXCLUDE=LOGIN.COM *.COM FRANCISSCOTT
```

3.2.4.6 /EXPIRED Qualifier

The /EXPIRED qualifier selects files for encryption according to the dates on which they expire. (The expiration date is set with the SET FILE /EXPIRATION_DATE command.) This qualifier is meaningful only when used with either the /BEFORE or the /SINCE qualifier. The /EXPIRED qualifier has the format:

/EXPIRED /BEFORE[=*time*]

or

/EXPIRED /SINCE[=*time*]

where *time* is an OpenVMS time.

For more information on time specifications, see the *OpenVMS DCL Dictionary*. If you do not specify a time, TODAY is used. TODAY is the current day, month, and year at 00:00:00.

The following command selects for encryption all files in the current directory matching the wildcard file specification of *.COM that expire after 00:00:00 15-APR-2001.

```
$ ENCRYPT /EXPIRED /SINCE=15-APR-2001 *.COM FRANCISSCOTT
```

Do not use the /EXPIRED qualifier with either the /BACKUP or the /MODIFIED qualifier.

3.2.4.7 /MODIFIED Qualifier

The /MODIFIED qualifier selects files for encryption according to the dates on which they were last modified. This qualifier is meaningful only when used with either the /BEFORE or the /SINCE qualifier. The /MODIFIED qualifier has the format:

/MODIFIED /BEFORE[=*time*]

or

/MODIFIED /SINCE[=*time*]

where

time is an OpenVMS time.

For more information on time specifications, see the *OpenVMS DCL Dictionary*. If you do not specify a time, TODAY is used. TODAY is the current day, month, and year at 00:00:00.

The following command selects for encryption all files in the current directory matching the wildcard file specification of *.COM that were modified after 00:00:00 15-APR-2001.

```
$ ENCRYPT /MODIFIED /SINCE=15-APR-2001 *.COM FRANCISSCOTT
```

Do not use the /MODIFIED qualifier with either the /BACKUP or the /EXPIRED qualifier.

3.2.4.8 /SINCE Qualifier

The /SINCE qualifier selects for encryption files that have a creation date after the time specified with the qualifier. The /SINCE qualifier has the format:

/SINCE[=*time*]

where

time is an OpenVMS time.

For more information on time specifications, see the *OpenVMS DCL Dictionary*. If you do not specify a time, TODAY is used. TODAY is the current day, month, and year at 00:00:00.

The following command selects for encryption all files in the current directory matching the wildcard file specification of *.COM that were created after 00:00:00 15-APR-2001.

```
$ ENCRYPT /SINCE=15-APR-2001 *.COM FRANCISSCOTT
```

3.2.5 Deleting Encrypted Files

By default, when the ENCRYPT software encrypts an input file and writes the resulting output file, the input file is retained. However, do not encrypt a file and then leave the plaintext file online if you are concerned about the security of the file.

You can use the DCL DELETE command with the /ERASE qualifier to remove the contents of the plaintext file from the disk, or you can use the following qualifiers with the ENCRYPT command:

- /DELETE
- /ERASE

3.2.5.1 /DELETE Qualifier

The /DELETE qualifier deletes the input file after the encryption operation completes and the output file is written and closed. If you have multiple versions of the input file, they are not all deleted. /DELETE acts on only the version of the input file that you encrypted.

To delete the unencrypted input file from the disk, use the /DELETE qualifier. The following command specifies that the SAVEDMAIL.MAI file be encrypted using the TWENTYFIVECENTS encryption key. Because the /DELETE qualifier is specified, the input file is deleted after the encrypted output file is written.

```
$ ENCRYPT /DELETE SAVEDMAIL.MAI TWENTYFIVECENTS
```

3.2.5.2 /ERASE Qualifier

When you delete or purge a file, the file's header record is destroyed so that the file can no longer be accessed by normal means. The information in the file, however, stays on the disk until it is overwritten. Disk scavenging is a technique used to obtain such file data from a disk. To thwart disk scavenging, use the /ERASE qualifier with the /DELETE qualifier. When you specify /ERASE, the OpenVMS operating system overwrites the location in which the input file was stored with the data security pattern. The data no longer exists.

The following command specifies that after SAVEDMAIL.MAI is encrypted, the input file is erased with the data security pattern before being deleted.

```
$ ENCRYPT /DELETE /ERASE SAVEDMAIL.MAI TWENTYFIVECENTS
```

3.2.6 Encryption Algorithms

Files are encrypted using a randomly generated data key. One benefit of this procedure is that two files identical in plaintext form and encrypted with the same command are not identical in their encrypted form.

The Encryption for OpenVMS implementation of DES uses the following modes of the DES algorithm:

- Cipher Block Chaining (DESCBC)
- Electronic Codebook (DESECB)
- Cipher Feedback (DESCFB)

These modes perform the encryption operation differently, as follows:

- DESCBC (default)
  1. Input is taken in 8-byte blocks.
  2. DESCBC performs an exclusive OR operation (XOR) on each block. (An XOR is a bit-by-bit modulo-2 addition without carrying. For example, the result of performing an XOR on the binary numbers 001 and 111 is 110.)

     The first XOR operation is performed on the first block of input and the initialization vector. (An initialization vector is used to start the chaining of data because there is no ciphertext to affect the encryption of the first block of data.)
  3. The resulting block is encrypted.
  4. The next XOR operation is performed on the resulting block of ciphertext and the next block of plaintext, and so on.
  5. If fewer than 8 bytes are left for the last iteration, the block is padded with bytes of arbitrary value.
  6. Each block of 8 bytes is encrypted under the same key value.
  7. The DESCBC algorithm is used to encrypt the data key and the initialization vector. The encrypted key and initialization vector are stored with the encrypted file. The DESCBC algorithm is also used by default to encrypt the file data.
- DESECB
  1. Input is taken in 8-byte blocks.
  2. If the input consists of less than 8 bytes, it is padded with nulls.

3. Each block is processed under the DES algorithm with the same key.
4. The result is an 8-byte block of output that is independent of all other blocks of output.

- DESCFB
  1. Input is taken as a series of 1-byte quantities.
  2. They are shifted to the left and concatenated with the results of previous iterations.
  3. DESCFB uses an initialization vector in the first iteration.
  4. Only the exact number of bytes specified in the input are used.
  5. The output byte count equals the input byte count (no padding).

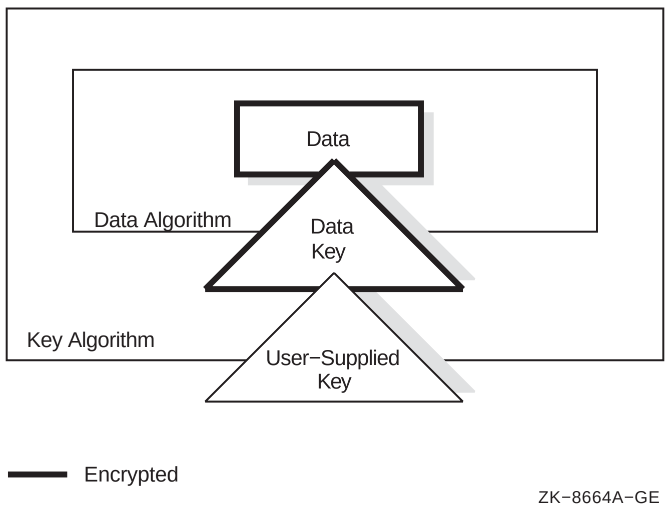
For details about the advantages of each mode, see one of the numerous texts available on this subject.

3.2.7 Encryption Algorithm Qualifiers

You can choose an encryption algorithm for encrypting either the data key or the file data. Figure 3–1 shows the relationship of encryption keys and algorithms. The figure shows that:

- To encrypt the key — use the /KEY_ALGORITHM qualifier to specify an algorithm other than DESCBC.
- To encrypt the file — use the /DATA_ALGORITHM qualifier to specify an algorithm other than DESCBC.

Figure 3–1 Relationship of Keys and Algorithms



The qualifier you use affects the correct decryption procedure:

- If you use the /DATA_ALGORITHM qualifier to encrypt, you do NOT need to specify this algorithm when you decrypt.
- If you use the /KEY_ALGORITHM qualifier to encrypt, you DO need to specify this algorithm when you decrypt.

3.2.7.1 /KEY_ALGORITHM Qualifier

To specify an algorithm other than DESCBC to encrypt the key and initialization vector, use the /KEY_ALGORITHM qualifier. This qualifier has the format:

/KEY_ALGORITHM={DESCBC (default) | DESECB | DESCFB}

For example, the following command uses the DESCFB algorithm with the TWENTYFIVECENTS key to protect the data key and the initialization vector.

```
$ ENCRYPT /KEY_ALGORITHM=DESCFB SAVEDMAIL.MAI TWENTYFIVECENTS
```

3.2.7.2 /DATA_ALGORITHM Qualifier

To specify an algorithm for encrypting files other than the default, use the /DATA_ALGORITHM qualifier. This qualifier has the format:

/DATA_ALGORITHM={DESCBC (default) | DESECB | DESCFB}

For example, the following command encrypts the SAVEDMAIL.MAI file using the Cipher Feedback mode of the DES algorithm (DESCFB).

```
$ ENCRYPT /DATA_ALGORITHM=DESCFB SAVEDMAIL.MAI TWENTYFIVECENTS
```

If you use the default value of DESCBC for the /DATA_ALGORITHM qualifier when encrypting a file, this qualifier is optional for decrypting the file.

3.2.8 File Compression

To reduce the size of the plaintext file before encrypting it, use the /COMPRESS qualifier. Data compression can save media space when physically transporting encrypted files and can save time when electronically transporting encrypted files across a network.

Compression efficiency depends on the structure of the data in your file. Evaluate a performance tradeoff when deciding whether or not to use this qualifier. Decryption is generally faster on a compressed file, but encryption takes longer. You might choose to use the /COMPRESS qualifier when the following conditions apply:

- The file will be decrypted many times.
- The file is at least 200 disk blocks in size.

The following command compresses the SAVEDMAIL.MAI file before encrypting it.

```
$ ENCRYPT /COMPRESS SAVEDMAIL.MAI TWENTYFIVECENTS
```

Note

If you use the /COMPRESS qualifier when encrypting a file, you need not specify this qualifier when decrypting the file. If necessary, the file is automatically decompressed when it is decrypted.

3.2.9 Displaying the Version Number

To identify the version of Encryption software running on your system, use the /VERSION qualifier. For example:

```
$ ENCRYPT /VERSION
Copyright (c) Compaq 2001, Digital Equipment Corporation. 1978, 1997. All
rights reserved.
Compaq Encryption V1.6)
```

3.3 Authenticating Files

Authentication is the checking of files to determine whether or not they have been modified.

The ENCRYPT /AUTHENTICATE command detects any modification of either plaintext or ciphertext files. The software calculates a Message Authentication Code (MAC) based on the contents of the files and associates it with one or more files. An additional MAC is created that is based on security settings unless you specifically request that the security MAC not be created. At a later time, when you want to check file integrity, the software recalculates the MACs and then compares the current and stored MACs. Before you use the ENCRYPT /AUTHENTICATE command, complete the process that associates MACs with files (see Section 3.3.1).

The ENCRYPT /AUTHENTICATE command has the following syntax:

ENCRYPT /AUTHENTICATE *file-spec key-name* [*qualifiers*]

where

file-spec	is the name of the file you want to check.
key-name	is the name of the key. Specify a 1-to-243-character string.
qualifiers	are options that control the encryption process or the selection of files you want to encrypt

A summary report on the authentication operation is displayed on SYS$OUTPUT.

The following qualifiers are valid with ENCRYPT /AUTHENTICATE:

- /[NO]DATABASE[=*file-spec*]

  Specifies a file in which to store binary MAC values created by using the file contents as input

- /LOG

  Displays the results of the authentication operation for each file

- /MULTIPLE_FILES

  Indicates that the *file-spec* parameter represents a list of file names to be checked

- /[NO]OUTPUT[=*file-spec*]

  Specifies a file in which to store readable MAC values

- /[NO]SECURITY[=*file-spec*]

  Generates a MAC using the file's security settings: owner, protection settings, and optional ACL, and specifies the file in which to store the binary MAC values.

- /[NO]UPDATE

  Associates new MAC values with one or more files

In addition, you can use all the file selection qualifiers available to the ENCRYPT command: /BACKUP, /BEFORE, /BY_OWNER, /CONFIRM, /EXCLUDE, /EXPIRED, /MODIFIED, and /SINCE (see Section 3.2.4).

The following sections describe how to use the /DATABASE, /LOG, /SECURITY, /OUTPUT, and /UPDATE qualifiers with ENCRYPT /AUTHENTICATE.

3.3.1 Associating MACs with Files

To associate MACs with a file or to replace former MAC values with new MAC values, use the /UPDATE qualifier. The /UPDATE qualifer updates two different MACs created from file contents and from security settings. The following command creates MAC values for all files in the current directory.

```
$ ENCRYPT /AUTHENTICATE *.* whitehen /UPDATE

%ENCRYPT-I-SUMMARY1,   Summary:   Files successfully authenticated: 0
%ENCRYPT-I-SUMMARY2,              Files failing authentication:  0
%ENCRYPT-I-SUMMARY3,              Files not in database:  3
%ENCRYPT-I-SECSUMM1,   Summary:   Security settings authenticated: 0
%ENCRYPT-I-SECSUMM2,              Security settings failing authentication: 0
%ENCRYPT-I-SECSUMM3,              Security settings not in database: 3
```

Two sets of summary information are displayed: the first set applies to the MAC values generated from the file contents, the second set applies to the MAC values generated from the security settings. Because this is the first time MACs are associated with these files, none are reported as authenticated (summary message 1 for each set) or as having failed authentication (summary message 2 for each set). The last message in each set reports that no previous MACs were associated with these files.

The MACs are stored in a binary database. Therefore, you cannot specify /NODATABASE or /NOSECURITY with /UPDATE. For more information, see Section 3.3.3.

3.3.2 Checking Files

With no other qualifiers, the ENCRYPT /AUTHENTICATE command compares previous MACs with current MACs. In addition, the software reports on files with no currently associated MACs.

The following command reports on the status of all the files in the current directory.

```
$ ENCRYPT /AUTHENTICATE *.* whitehen

%ENCRYPT-I-NOUPDATE, database will not be updated with new authentication codes
%ENCRYPT-I-SUMMARY1, Summary:   Files successfully authenticated: 3
%ENCRYPT-I-SUMMARY2,            Files failing authentication:  0
%ENCRYPT-I-SUMMARY3,            Files not in database:  0
%ENCYRPT-I-SECSUMM1, Summary:   Security settings authenticated: 3
%ENCYRPT-I-SECSUMM2,            Security settings failing authenticated: 0
%ENCYRPT-I-SECSUMM3,            Security settings not in database:0
```

3.3.3 Specifying a File for MACs Generated from File Contents

A database file stores MAC values in binary format. By default, binary MAC values created from the file contents are stored in SYS$LOGIN:ENCRYPT$MAC.DAT. You can use the /DATABASE qualifier to store the MAC values in an alternate file.

The following command selects an alternate file in which to store the MAC values.

```
$ ENCRYPT /AUTHENTICATE *.com whitehen /DATABASE=[MACS]MACCHECK.DAT /UPDATE

%ENCRYPT-I-NEWDB,     New authentication code database has been created
%ENCRYPT-I-SUMMARY1,  Summary: Files successfully authenticated: 0
%ENCRYPT-I-SUMMARY2,  Files failing authentication:  0
%ENCRYPT-I-SUMMARY3,  Files not in database:  6
```

When you specify /NODATABASE, the MAC values are not stored. The next time you use the ENCRYPT /AUTHENTICATE command, the files are treated as new since there are no current MAC values to check.

3.3.4 Specifying a Security MAC File

MAC entries based on security settings are automatically generated and stored in a security database when the /UPDATE qualifier is used (see Section 3.3.1). If you do not want to generate a MAC value based on security settings, use the /NOSECURITY qualifier on the ENCRYPT /AUTHENTICATE command line.

The entries in the security database are generated by using the security settings: owner, protection settings, and an ACL if one is associated with the file. By default, security MAC values are stored in the database ENCRYPT$SEC.DAT. You can use the /SECURITY qualifier to store security MAC values in an alternate file.

The following command selects an alternate file in which to store security MAC values.

```
$ ENCRYPT /AUTHENTICATE *.com seveneleven /SECURITY=SECURITYMAC.DAT /UPDATE

%ENCYRPT-I-NEWSECDB, New authentication security settings database has been created
%ENCRYPT-I-SUMMARY1, Summary:   Files successfully authenticated: 0
%ENCRYPT-I-SUMMARY2,            Files failing authentication: 0
%ENCRYPT-I-SUMMARY3,            Files not in database: 3
%ENCRYPT-I-SECSUMM1, Summary:   Security settings authenticated: 0
%ENCRYPT-I-SECSUMM2,            Security settings failing authentication: 0
%ENCRYPT-I-SECSUMM3,            Security settings not in database: 3
```

3.3.5 Specifying a Listing File

In addition to a binary MAC database, Encryption stores MAC values and status information in readable form. By default, readable MAC values are stored in SYS$LOGIN:ENCRYPT$MAC.LIS.

To store readable values in an alternate file, use the /OUTPUT qualifier. The file extension defaults to .LIS. For example, this command specifies SYS$LOGIN:08MAC.LIS as the listing file:

```
$ ENCRYPT /AUTHENTICATE *.*  whitehen /OUTPUT=08MAC

%ENCRYPT-I-NOUPDATE, database will not be updated with new authentication codes
%ENCRYPT-I-SUMMARY1, Summary: Files successfully authenticated: 6
%ENCRYPT-I-SUMMARY2, Files failing authentication:  0
%ENCRYPT-I-SUMMARY3, Files not in database:  0
```

To display the listing on SYS$OUTPUT, enter:

```
$ TYPE 08MAC.LIS

File Integrity Report  22-APR-2001 10:50:22.62       Compaq Encryption  V1.6  Page  1
Authentication database: DISK_1:[000000.SCRATCH]ENCRYPT$MAC.DAT;1

File name                          Stored MAC         Current MAC  Status
=================================  ================  ===========  ======
DISK_1[SCRATCH]EXAMPLE.FILE;1      90E70CB4E8E96BBF   (same)
 owner: [1,1]  prot: (RWED, RWED, RWED, )
DISK_1[SCRATCH]PICTURE.SLS;1       FCAD115A72E7934A   (same)
 owner: [1,1]  prot: (RWED, RWED, RWED, )
DISK_1[SCRATCH]RELEASE.TXT;1       11375BD8D504ABB3   (same)
 owner: [1,1]  prot: (RWED, RWED, RWED, )
DISK_1[SCRATCH]RELEASE_NOTES.PS;3  2632027C133A8B5F   (same)
 owner: [1,1]  prot: (RWED, RWED, RWED, )
DISK_1[SCRATCH]SCHEDULE.LIST;3     852D440358FBFF95   (same)
 owner: [1,1]  prot: (RWED, RWED, RWED, )
DISK_1[SCRATCH]WATCH_MAIL.COM;5    B75D00EC4991662C   (same)
 owner: [1,1]  prot: (RWED, RWED, RWED, )

Summary:        Files successfully authenticated: 6
                Files failing authentication: 0
                Files not in database: 0

Summary:        Security settings authenticated: 6
                Security settings failing authentication: 0
                Security settings not in database: 0
```

To suppress the creation of this listing, use the /NOOUTPUT qualifier.

3.3.6 Logging the Authentication Operation

To display the results of the authentication operation on each file, use the /LOG qualifier. For example, the following command displays the results of each file authentication on your terminal screen.

```
$ ENCRYPT /AUTHENTICATE /LOG *.* whitehen

%ENCRYPT-I-NOUPDATE, database will not be updated with new authentication codes
%ENCRYPT-S-AUTHMATCH, File DISK_1:[SCRATCH]EXAMPLE.TXT;1 successfully authenticated
%ENCRYPT-S-SECAUTHMATCH, Security settings for DISK_1:[SCRATCH]EXAMPLE.TXT successfully authenticated
%ENCRYPT-S-AUTHMATCH, File DISK_1:[SCRATCH]TEST.TXT;1 successfully authenticated.
%ENCRYPT-S-SECAUTHMATCH, Security settings for DISK_1:[SCRATCH]TEST.TXT successfully authenticated
%ENCRYPT-S-AUTHMATCH, File DISK_1:[SCRATCH]RELEASE.TXT;2 successfully authenticated.
%ENCRYPT-S-SECAUTHMATCH, Security settings for DISK_1:[SCRATCH]RELEASE.TXT successfully authenticated
%ENCRYPT-I-SUMMARY1, Summary:   Files successfully authenticated: 6
%ENCRYPT-I-SUMMARY2,            Files failing authentication:0
%ENCRYPT-I-SUMMARY3,            Files not in database:0

%ENCRYPT-I-SECSUMM1, Summary:   Security settings authenticated: 6
%ENCRYPT-I-SECSUMM2,            Security settings failing authentication:0
%ENCRYPT-I-SECSUMM3,            Security settings not in database:0
```

3.4 Deleting Key Definitions

When a key outlives its usefulness, delete it from a key storage table. Enter the ENCRYPT /REMOVE_KEY command and specify the name under which the encrypted key value was stored in the key table. The key name is the character string previously defined with an ENCRYPT /CREATE_KEY command.

The ENCRYPT /REMOVE_KEY command has the following format:

ENCRYPT /REMOVE_KEY *key-name* [*qualifiers*]

By default, the ENCRYPT /REMOVE_KEY command deletes the key definition from the process key storage table. Logging out a process also removes a key definition from the process key storage table.

To remove a key definition from the job, group, or system storage table, specify the /JOB, /GROUP, or /SYSTEM qualifier with the ENCRYPT /REMOVE_KEY command. Just as you need privileges to create group or system keys, you need privileges to delete them.

For example, the following command deletes the HAMLET key from the system key storage table:

```
$ ENCRYPT /REMOVE_KEY HAMLET /SYSTEM
```

To verify key removal, use the /LOG qualifier with the ENCRYPT /REMOVE_KEY command. The following command reports that the key HAMLET is removed:

```
$ ENCRYPT /REMOVE_KEY HAMLET /SYSTEM /LOG
%ENCRYPT-S-KEYDEL, key deleted for key name = HAMLET
```

3.5 Decrypting Files

To gain access to the data in an encrypted file, decrypt the file using the DECRYPT command. Follow these steps:

1. Specify the same key used to encrypt the file.

   See if you need to redefine the key using the ENCRYPT /CREATE_KEY command. For example, if the key was in the process key storage table and the process logged out, the key is no longer defined.
2. Specify the algorithm with the /KEY_ALGORITHM qualifier, if you did not encrypt the file with the default algorithm.

The DECRYPT command has the following format:

DECRYPT *file-spec key-name* [*qualifiers*]

where

file-spec	is the name of the file.
key-name	is the name of the key.
qualifiers	are options that control the decryption process or the selection of files you want to decrypt

3.5.1 Input File Specification

For the ciphertext file, which is the file to be decrypted, specify a file that resides on disk and that is not a directory file.

To specify multiple input files to the DECRYPT command, use wildcard characters in the file specification. To control file selection, specify the appropriate DECRYPT command qualifiers (see Section 3.5.4). Do not use wildcard characters to specify directory files or files containing bad blocks.

3.5.2 Output File Specification

The result of the decryption operation is a plaintext file. One plaintext file is created for each input file that is decrypted. By default, the DECRYPT command writes each plaintext file to a separate output file with a file specification that defaults to the input file specification with a version number that is one higher than that of the input file.

You can specify an alternate output file specification with the /OUTPUT qualifier. When specifying the /OUTPUT qualifier, you specify those parts of the file specification that you want to be different from the defaults. You do not need to specify an entire file specification; any fields omitted in the file specification default to the input file specification.

For example, the following DCL command selects for decryption all files in the current directory matching the wildcard file specification of *.ENC. The /OUTPUT qualifier specifies that any output files created have a file type of COM.

```
$ DECRYPT *.ENC/OUTPUT=.COM FRANCISSCOTT
```

3.5.3 Displaying Processing Information

By default, information about the decryption operation is not displayed on SYS$COMMAND. To display this information, use the /SHOW qualifier. The /SHOW qualifier has the format:

/SHOW=*keyword*

or

/SHOW=(*keyword-list*)

Specify one or more of the following keywords:

- FILES (Section 3.5.3.1)
- STATISTICS (Section 3.5.3.2)

3.5.3.1 FILES Keyword

Use the FILES keyword to display the input and output file specifications as decryption proceeds. For example, /SHOW=FILES in the following command specifies that each input and output file specification be displayed as it is decrypted.

```
$ DECRYPT /SHOW=FILES *.COM FRANCISSCOTT

%ENCRYPT-S-DECRYPTED, DISK2:[FLYNN]MOVE.COM.3 decrypted to
  DISK2:[FLYNN]MOVE.COM;4 (8 blocks)
.
.
.
```

3.5.3.2 STATISTICS Keyword

Use the STATISTICS keyword to display encryption stream statistics after the completion of each file decryption operation. The statistics displayed are:

Bytes processed
Internal records processed
CPU time consumed within the encryption algorithm

The following command specifies that the decryption stream statistics be displayed on SYS$COMMAND.

```
$ DECRYPT /SHOW=STATISTICS *.COM FRANCISSCOTT

%ENCRYPT-S-STATISTICS, encryption stream statistics:
         Total Records: 65
         Total Bytes: 4083
         Total Time: 00:00:00:01.63
.
.
.
```

3.5.4 Specifying Files to Decrypt

You can use the DECRYPT command to specify multiple input files by using wildcard characters in the input file specification. The command also provides the following qualifiers for selecting files:

- /BACKUP
- /BEFORE
- /BY_OWNER
- /CONFIRM
- /EXCLUDE
- /EXPIRED
- /MODIFIED
- /SINCE

The following sections describe these qualifiers.

3.5.4.1 /BACKUP Qualifier

The /BACKUP qualifier selects files for decryption according to the date of their most recent backup. This qualifier is meaningful only when used with either the /BEFORE or the /SINCE qualifier. The /BACKUP qualifier has the format:

/BACKUP /BEFORE[=*time*]

/BACKUP /SINCE[=*time*]

where

time is an OpenVMS time.

For more information on time specifications, see the *OpenVMS DCL Dictionary*. If you do not specify a time, TODAY is used. TODAY is the current day, month, and year at 00:00:00.

The following command selects for decryption all files in the current directory matching the wildcard file specification of *.COM that had backup copies made before 00:00:00 15-APR-2001.

```
$ DECRYPT /BACKUP /BEFORE=15-APR-2001 *.COM FRANCISSCOTT
```

Do not use the /BACKUP qualifier with either the /EXPIRED or the /MODIFIED qualifier.

3.5.4.2 /BEFORE Qualifier

The /BEFORE qualifier selects files for decryption that have a creation date before the time specified with the qualifier. The /BEFORE qualifier has the format:

/BEFORE[=*time*]

where

time is an OpenVMS time.

For more information on time specifications, see the *OpenVMS DCL Dictionary*. If you do not specify a time, TODAY is used. TODAY is the current day, month, and year at 00:00:00.

The following command selects for decryption all files in the current directory matching the wildcard file specification of *.COM that were created before 00:00:00 15-APR-2001.

```
$ DECRYPT /BEFORE=15-APR-2001 *.COM FRANCISSCOTT
```

3.5.4.3 /BY_OWNER Qualifier

Use the /BY_OWNER qualifier to select files for decryption that have a particular owner User Identification Code (UIC). If no UIC is specified with the qualifier, the UIC of the current process is used. The /BY_OWNER qualifier has the format:

/BY_OWNER=*uic*

where

uic is the UIC of the owner of the file.

For more information on UIC format, see the *OpenVMS DCL Dictionary*. The following command selects for decryption all files in the current directory owned by the user whose UIC is [FLYNN] that match the wildcard file specification of *.COM.

```
$ DECRYPT /BY_OWNER=[FLYNN] *.COM FRANCISSCOTT
```

3.5.4.4 /CONFIRM Qualifier

By default, all input files specified on the command line are processed without confirming that each file is selected for decryption. Use the /CONFIRM qualifier if you want a prompt with the name of each file selected for decryption. Your response controls whether or not a particular file is decrypted.

You can choose any of the following responses:

Response	Meaning
YES	Decrypt the file.
NO or Return	Do not decrypt the file. This is the default.
QUIT or Ctrl/Z	Do not decrypt the file or any subsequent files.
ALL	Decrypt the file and all subsequent files.

The following command selects all files in the current directory matching the wildcard file specification of *.COM for decryption. Because the /CONFIRM qualifier is specified, the user is prompted on a file-by-file basis to confirm that each file is to be decrypted. Because the prompt is answered in the affirmative for the file MOVE.COM;3, the output file MOVE.COM;4 is created.

```
$ DECRYPT /CONFIRM *.COM FRANCISSCOTT
Decrypt DISK2:[FLYNN]MOVE.COM;3 ? [N] YES
```

3.5.4.5 /EXCLUDE Qualifier

Use the /EXCLUDE qualifier to exclude one or more files from a decryption operation. If a file matches the file specification provided with the qualifier, the file is not decrypted. The /EXCLUDE qualifier has the format:

/EXCLUDE=(*file-spec*[,...])

where

file-spec is the file specification of the file to remain encrypted.

When specifying only one file, you can omit the parentheses. Wildcard characters are allowed in the file specification. With the /EXCLUDE qualifier, there is no default for the file specification.

Since directory files are never encrypted, you need not specify them with the /EXCLUDE qualifier. However, if you do specify `/EXCLUDE=*.DIR`, you will not get the warning message `%ENCRYPT-W-FILNODIR, file encryption of directories is not supported, filename.dir`.

The following command selects for decryption all files in the current directory that match the wildcard file specification of *.COM, except LOGIN.COM, which is specified with /EXCLUDE.

```
$ DECRYPT /EXCLUDE=LOGIN.COM *.COM FRANCISSCOTT
```

3.5.4.6 /EXPIRED Qualifier

The /EXPIRED qualifier selects files for decryption according to the dates on which they expire. (The expiration date is set with the SET FILE/EXPIRATION_DATE command.) This qualifier is meaningful only when used with either the /BEFORE or the /SINCE qualifier. The /EXPIRED qualifier has the format:

/EXPIRED /BEFORE[=*time*] /EXPIRED /SINCE[=*time*]

where

time is an OpenVMS time.

For more information on time specifications, see the *OpenVMS DCL Dictionary*. If you do not specify a time, TODAY is used. TODAY is the current day, month, and year at 00:00:00.

The following command selects for decryption all files in the current directory matching the wildcard file specification of *.COM that expire after 00:00:00 15-APR-2001.

```
$ DECRYPT /EXPIRED /SINCE=15-APR-2001 *.COM FRANCISSCOTT
```

Do not use the /EXPIRED qualifier with either the /BACKUP or the /MODIFIED qualifier.

3.5.4.7 /MODIFIED Qualifier

The /MODIFIED qualifier selects files for decryption according to the dates on which they were last modified. This qualifier is meaningful only when used with either the /BEFORE or the /SINCE qualifier. The /MODIFIED qualifier has the format:

/MODIFIED /BEFORE[=*time*] /MODIFIED /SINCE[=*time*]

where

time is an OpenVMS time.

For more information on time specifications, see the *OpenVMS DCL Dictionary*. If you do not specify a time, TODAY is used. TODAY is the current day, month, and year at 00:00:00.

The following command selects for decryption all files in the current directory matching the wildcard file specification of *.COM that were modified after 00:00:00 15-APR-2001.

```
$ DECRYPT /MODIFIED /SINCE=15-APR-2001 *.COM FRANCISSCOTT
```

Do not use the /MODIFIED qualifier with either the /BACKUP or the /EXPIRE qualifier.

3.5.4.8 /SINCE Qualifier

The /SINCE qualifier selects files for decryption that have a creation date after the time specified with the qualifier. The /SINCE qualifier has the format:

/SINCE[=*time*]

where

time is an OpenVMS time.

For more information on time specifications, see the *OpenVMS DCL Dictionary*. If you do not specify a time, TODAY is used. TODAY is the current day, month, and year at 00:00:00.

The following command selects for decryption all files in the current directory matching the wildcard file specification of *.COM that were created after 00:00:00 15-APR-2001.

```
$ DECRYPT /SINCE=15-APR-2001 *.COM FRANCISSCOTT
```

3.5.5 Deleting Decrypted Files

By default, the input file is retained after a file is decrypted and written to the resulting output file. To save space, after you have decrypted a file, you may want to remove the encrypted file from your disk.

You can use the DCL DELETE command with the /ERASE qualifier to remove the contents of the file from the disk, or you can use the /DELETE and /ERASE qualifiers with the DECRYPT command.

3.5.5.1 /DELETE Qualifier

The /DELETE qualifier deletes the input file after the decryption operation completes and the output file is written and closed. If you have multiple versions of the input file, they are not all deleted. /DELETE acts on only the version of the input file that you encrypted.

The following command specifies that the SAVEDMAIL.MAI file be decrypted using the TWENTYFIVECENTS encryption key. Because the /DELETE qualifier is specified, the input file is deleted after the output file is written.

```
$ DECRYPT /DELETE SAVEDMAIL.MAI TWENTYFIVECENTS
```

3.5.5.2 /ERASE Qualifier

To prevent disk scavenging, use the /ERASE qualifier with the /DELETE qualifier. For example, the following command decrypts the SAVEDMAIL.MAI file using the TWENTYFIVECENTS encryption key, erases the input file with the data security pattern, and deletes the file.

```
$ DECRYPT /DELETE /ERASE SAVEDMAIL.MAI TWENTYFIVECENTS
```

With the following command, the SAVEDMAIL.MAI file is decrypted using the TWENTYFIVECENTS encryption key, but the input file is not erased with the data security pattern before being deleted.

```
$ DECRYPT /DELETE /NOERASE SAVEDMAIL.MAI TWENTYFIVECENTS
```

3.5.6 Algorithm Qualifiers

The algorithm qualifier you use to encrypt determines the correct decryption procedure:

- If you use the /DATA_ALGORITHM qualifier to encrypt, do not specify this algorithm when you decrypt.
- If you use the /KEY_ALGORITHM qualifier to encrypt, specify this algorithm when you decrypt.

The /KEY_ALGORITHM qualifier has the format:

/KEY_ALGORITHM=*algorithm*

where

algorithm is one of the following values:

- DESCBC (the default)
- DESECB
- DESCFB

For example, if SAVEDMAIL.MAI is encrypted with /KEY_ALGORITHM=DESCFB, decrypt the file with the same /KEY_ALGORITHM=DESCFB qualifier, as follows:

```
$ ENCRYPT /KEY_ALGORITHM=DESCFB SAVEDMAIL.MAI TWENTYFIVECENTS
$ DECRYPT /KEY_ALGORITHM=DESCFB SAVEDMAIL.MAI TWENTYFIVECENTS
```

3.6 Encrypting Save Sets

The OpenVMS BACKUP utility provides protection against file or volume corruption by creating functionally equivalent backup copies. Files created by BACKUP are called **save sets** and are written in BACKUP format so that only BACKUP can interpret the data in a save set. See the *OpenVMS System Management Utilities Reference Manual* for more information on the BACKUP utility. When you create save sets, you can also encrypt them by using the BACKUP /ENCRYPT command.

Note

Standalone BACKUP, which is a version of the BACKUP utility that runs without the support of the OpenVMS operating system, does not support the /ENCRYPT qualifier.

BACKUP /ENCRYPT requires a key. All the files in the save set are encrypted under the same key. When you use the /ENCRYPT qualifier to specify a write operation for an encrypted save set, the BACKUP utility creates a key by generating a 16-byte random number from the time of day and other transient data. To make this random number even more random, BACKUP encrypts this 16-byte value once using itself as a key with the DESCBC algorithm. The first eight bytes of the result are used as the encrypting key for the save set, and the second eight bytes are used as the initialization vector for the context area.

One benefit of this procedure is that two save sets created with the same command from the same set of files are not identical in their encrypted form.

You can override the system-generated encrypting key and initialization vector by issuing either of the following commands:

- ENCRYPT /CREATE_KEY
- BACKUP /ENCRYPT=(VALUE=*key-value*)

For greater security, specify the /ENCRYPT qualifier with no parameters. The software prompts you for a key value. When you enter it, the software does not echo what you type and, for verification, prompts you to retype the value.

If you define a key with the ENCRYPT /CREATE_KEY command, specify that key name on the BACKUP command line with the /ENCRYPT=(NAME=*key-name*) qualifier.

By default, BACKUP encrypts save set data using the DESCBC algorithm. The key and algorithm you specify to override the defaults are used to encrypt only the data key and the initialization vector.

BACKUP places the result of the encryption operation in the save set as a BACKUP attribute subrecord of the BACKUP summary record. At the time of a save set restore or listing operation, BACKUP uses the system-generated key or the key you supplied to decrypt the data key and the initialization vector value.

The BACKUP command qualifier /SAVE_SET is both an input save set qualifier and an output save set qualifier, as follows:

- When you specify the /SAVE_SET and /ENCYRPT qualifiers with an output save set specification, BACKUP writes file data (including file names and attributes) in an encrypted form into the save set.

- When you specify /SAVE_SET with an input save set specification, BACKUP uses the decryption key specified to access the file name, attributes, and data from the save set records. The ENCRYPT option decrypts the data files after BACKUP reads the data files from the save set medium and processes them according to the remaining qualifiers of the BACKUP command.

The following example creates an encrypted BACKUP file of the default directory, as follows:

1. ENCRYPT /CREATE_KEY defines a key, SANFRANCISCO, with this value: `A city set on a hill cannot be hid`.
2. BACKUP /ENCRYPT saves all the files in the default directory in a save set named 28JULSAVE.BCK and encrypts the save set.

   On device MKA600:, the data used to encrypt the file names, attributes, and all the other file data are encrypted with the default encryption algorithm DESCBC. The process uses the key defined as SANFRANCISCO.

```
$ ENCRYPT /CREATE_KEY SANFRANCISCO "A city set on a hill cannot be hid"
$ BACKUP /ENCRYPT=(NAME=SANFRANCISCO) * MKA600:28JULSAVE.BCK /SAVE_SET
```

The following example creates a save set of the latest version of all the files on a disk. The save set is encrypted using the DESCFB algorithm and the key value `Make peace`.

```
$ BACKUP /ENCRYPT=(VALUE="Make peace",ALGORITHM=DESCFB) *.* 28JULSAVE /SAVE_SET
```

3.6.1 Restoring Files

When you encrypt a save set, BACKUP does not store the information within the save set. Consequently, to decrypt an encrypted save set, specify /ENCRYPT with the RESTORE command so that BACKUP searches for the data encryption control record.

If you restore an unencrypted save set and mistakenly specify /ENCRYPT, BACKUP ignores the incorrect qualifier. If you try to restore an encrypted saveset without the /ENCYRPT qualifier or with a key name, you get the error message:

```
%BACKUP-F-ENCSAVSET, save set is encrypted, /ENCRYPT must be specified
```

The following commands restore file SALARY.DAT from a save set created with a BACKUP /ENCRYPT command:

```
$ ENCRYPT /CREATE_KEY CASTERBRIDGE "And all her shining keys"
$ BACKUP /ENCRYPT=(NAME=CASTERBRIDGE)
_$ From: MKA600:28JULSAVE.BCK /SELECT=SALARY.DAT
_$ To: SALARY28J.DAT
```

BACKUP tries to decrypt an encrypted save set by:

1. Decrypting the encryption data that was saved in an attribute subrecord.
2. Comparing a 32-bit checksum of the decrypted data key with the stored value.
3. If there is a match, BACKUP assumes the data key is valid and restores the save set.

4. If BACKUP finds a mismatch, which is likely if the data key or algorithm you specified in the BACKUP command is incorrect, the utility displays:

```
%BACKUP-F-ENCKEYMAT, the supplied decryption key does not yield a readable save set
```

3.6.2 Encrypting Distribution Files

BACKUP /ENCRYPT can create a distribution disc that is useful only to a customer who has the key used to encrypt the save sets in the distribution kit.

In the following example, three keys are defined with ENCRYPT /CREATE_KEY commands. With each of these keys, a software distribution disc is created with each product encrypted into its respective save set under a unique key.

```
$ ENCRYPT /CREATE_KEY SDXKEY "SDX V9.0 kit 99804034671838302"
$ BACKUP /ENCRYPT=(NAME=SDXKEY) /REWIND -
_From: MASTER:[SDXKIT]*.* MKA600:SDXKIT /SAVE_SET

$ ENCRYPT /CREATE_KEY RQPKEY "RQP V4.5 kit FWTEBCJDITROEMMKAZXRYTC"
$ BACKUP /ENCRYPT=(NAME=RQPKEY) -
_From: MASTER:[RQPKIT]*.* MKA600:RQPKIT /SAVE_SET

$ ENCRYPT /CREATE_KEY WOLKEY "WOL V2.0 kit 28374UEJDTLHGD84JF849SK95KD0"
$ BACKUP /ENCRYPT=(NAME=WOLKEY) -
_From: MASTER:[WOLKIT]*.* MKA600:WOLKIT /SAVE_SET
```

The resulting save sets can be restored on a customer's system only if the customer has received the appropriate key by licensing arrangement.

For example, the following commands restore save set WOLKIT:

```
$ ENCRYPT /CREATE_KEY WOLKEY "WOL V2.0 kit 28374UEJDTLHGD84JF849SK95KD0"
$ BACKUP /ENCRYPT=(NAME=WOLKEY) MKA600:WOLKIT /SAVE_SET SYSTEM:[RQPKIT]*.*
```

In the following example, the save set SDXKIT is restored without typing the key name and key value on the command line. Instead, the BACKUP /ENCRYPT command prompts for this information, which is not echoed on your screen.

```
$ BACKUP /ENCRYPT /REWIND MKA600:SDXKIT /SAVE_SET SYSTEM:[SDXKIT]*.*
Enter Key Value: (input not echoed)
Verify: (input not echoed)
```

4 Programming with Encryption for OpenVMS Routines

To program encryption operations into applications, use the Encryption for OpenVMS callable routines. Encryption provides the following routines, listed by function:

- Defining, generating, and deleting keys (Section 4.2):
  - ENCRYPT$DEFINE_KEY
  - ENCRYPT$GENERATE_KEY
  - ENCRYPT$DELETE_KEY
- Encrypting and decrypting files (Section 4.3):
  - ENCRYPT$ENCRYPT
  - ENCRYPT$ENCRYPT_FILE
  - ENCRYPT$DECRYPT
- Encrypting and decrypting records (Section 4.4):
  - ENCRYPT$DECRYPT_ONE_RECORD
  - ENCRYPT$ENCRYPT_ONE_RECORD
- Intializing and terminating the context area (Section 4.4):
  - ENCRYPT$INIT
  - ENCRYPT$FINI
- Returning statistics (Section 4.4):
  - ENCRYPT$STATISTICS

Installation Note

To use the programming interface from an image that was installed with privileges, ensure that the system startup procedure installs the ENCRYPSHR shareable image as a known image (see Section 2.4).

4.1 How the Routines Work

You can call the Encryption for OpenVMS routines from any language that supports the OpenVMS Calling Standard. After it is called, each routine:

- Performs its function
- Returns a 32-bit status code value for the calling program to determine success or failure
- Returns control to the calling program

The callable routines do not provide all the options of the file selection qualifiers available with the DCL ENCRYPT and DECRYPT commands. The functions of /BACKUP, /BEFORE, /BY_OWNER, /CONFIRM, /EXCLUDE, /EXPIRED, /SINCE, and /SHOW are supported only at the DCL-interface level. For more information, see the *Guide to Creating OpenVMS Modular Procedures*.

4.1.1 DES Key and Data Semantics

The NBS document FIPS-PUB-46 describes the operation of the DES algorithm in detail. The bit-numbering conventions in the NBS document are different from OpenVMS numbering conventions.

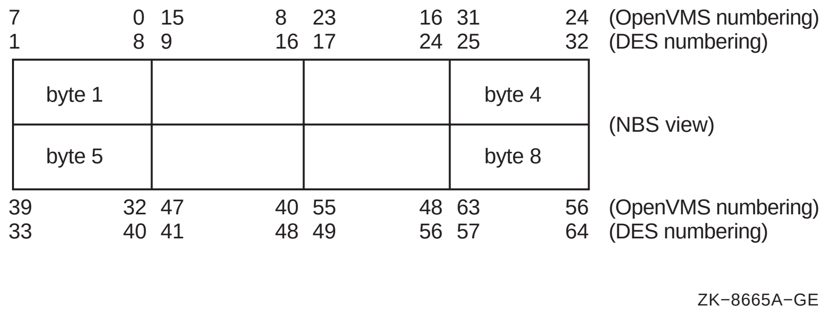
If you are using Encryption for OpenVMS routines in conjunction with an independently developed DES encryption system, ensure that you are familiar with the relationship between the NBS and OpenVMS numbering conventions. Table 4–1 highlights the differences.

Table 4–1 Comparison of NBS and OpenVMS Numbering Conventions

NBS	Encryption for OpenVMS
Numbers bits from left to right.	Numbers bits from right to left.
Displays bytes in memory from left to right.	Displays bytes in memory from right to left.
Handles keys and data in 8-byte blocks (see Figure 4–1).	Handles 8-byte blocks in OpenVMS display order (see Figure 4–2).
Treats keys and data as byte strings.	Treats keys and data as character strings.
The "most significant byte" is byte 1.	Same.
In DES keys, the parity bits are DES bits 8, 16, 24, and so forth.	In DES keys, the parity bits are OpenVMS bits 0, 8, 16, and so forth.
DES keys, when expressed as strings of hexadecimal digits, are given starting with the high digit of byte 1, then the low digit of byte 1, then the high digit of byte 2, and so forth, through the low digit of byte 8.	Same.

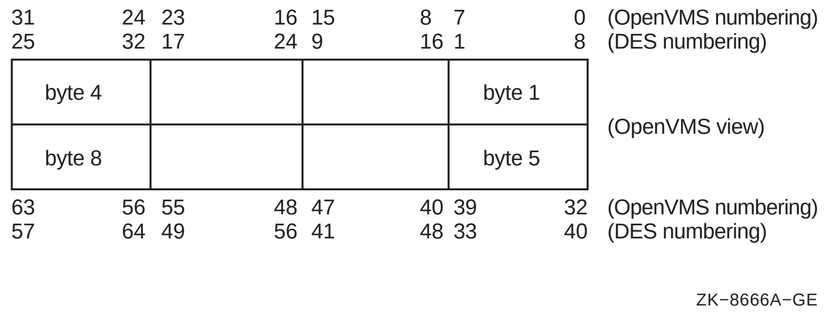
To convert a hexadecimal key string into the 8-byte binary key, convert from hex to binary one byte at a time. For example, a quadword hex-to-binary conversion, using the library subroutine OTS$CVT_TZ_L, yields an incorrect, byte-reversed key.

Figure 4–1 OpenVMS Numbering Overlay on FIPS-46 Numbering



7 0 15 8 23 16 31 24 (OpenVMS numbering)
1 8 9 16 17 24 25 32 (DES numbering)
byte 1
byte 4
(NBS view)
byte 5
byte 8
39 32 47 40 55 48 63 56 (OpenVMS numbering)
33 40 41 48 49 56 57 64 (DES numbering)
ZK−8665A−GE

Figure 4–2 NBS Numbering Overlay on an OpenVMS Quadword



31 24 23 16 15 8 7 0 (OpenVMS numbering)
25 32 17 24 9 16 1 8 (DES numbering)
byte 4
byte 1
(OpenVMS view)
byte 8
byte 5
63 56 55 48 47 40 39 32 (OpenVMS numbering)
57 64 49 56 41 48 33 40 (DES numbering)
ZK−8666A−GE

4.2 Maintaining Keys

When you use the Encryption routines, first define the key that will be used in the encryption operation. Similarly, to decrypt a file specify the same key. Table 4–2 describes the callable routines that maintain keys.

Table 4–2 Routines for Maintaining Keys

Routine	Description
ENCRYPT$DEFINE_KEY	Creates a key definition with a key name and a key value. Puts the definition into a key storage table. Similar to the ENCRYPT /CREATE_KEY command.
ENCRYPT$DELETE_KEY	Removes a key definition from a key storage table. Uses the key name to identify the key to be removed. Similar to the ENCRYPT /REMOVE_KEY command.
ENCRYPT$GENERATE_KEY	Generates random key values.

When you call these routines, use the following arguments:

- With ENCRYPT$DEFINE_KEY
  - To pass the values for the key name and key value, use the **key-name** and the **key-value** arguments.
  - To specify a key storage table, use the **key-flags** argument.

  - To specify other key options, use the **key-flags** argument.
  - To override key compression, use the **key-flags** argument.
- With ENCRYPT$DELETE_KEY
  - To pass the key name, use the **key-name** argument.
  - To specify the key storage table in which the key resides, use the **key-flags** argument.
- With ENCRYPT$GENERATE_KEY
  - To define the length of the key, use the **key-length** argument.
  - To specify the buffer into which the generated key is to be placed, use the **key-buffer** argument.
  - To specify the algorithm that will use the key, use the **algorithm-name** argument.
  - To optionally pass three arbitrary values for added security, use the **factor-a**, **factor-b**, and **factor-c** arguments. These values are randomizing factors when the routine generates a key value. For example, the factors might be:
    - Time an operation started
    - Size of a certain stack
    - Copy of the last command line

4.3 Operations on Files

The ENCRYPT$ENCRYPT_FILE routine is similar to the DCL ENCRYPT and DECRYPT commands in that you use this routine with entire files.

The ENCRYPT$ENCRYPT_FILE routine specifies the key, the input file specification, the output file specification, and other file operation information.

Specify the type of operation, either encryption or decryption, with the **file-flags** argument.

ENCRYPT$ENCRYPT_FILE does not require a prior call to ENCRYPT$INIT.

4.4 Operations on Records and Blocks

To operate on small records or blocks of data, use the following routines:

- ENCRYPT$ENCRYPT_ONE_RECORD
- ENCRYPT$DECRYPT_ONE_RECORD

These routines are a shorthand form of the ENCRYPT$INIT, ENCRYPT$ENCRYPT, ENCRYPT$DECRYPT, ENCRYPT$FINI sequence of calls.

Do not use these routines for data larger than a few records.

4.5 Routine Descriptions

This section describes the syntax of each callable routine. The routines are listed alphabetically.

4.5.1 Specifying Arguments

Each routine's argument list shows the mandatory arguments first, followed by the optional arguments. Brackets ([ ]) identify optional arguments in the argument list.

For example, this format line shows that the required arguments are **context**, **input**, and **output**, and that the optional arguments are **output-length** and **p1**:

ENCRYPT$DECRYPT context ,input ,output [,output-length] [,p1]

When you specify arguments, follow these guidelines:

- The order is important. Specify arguments in the order in which they appear in the argument list.
- Separate each argument with a comma.
- Pass a zero value for each optional argument that you omit.

4.5.2 Bitmasks

Constants are associated with the symbolic names of the bitmasks used by the Encryption routines. These constants are defined in the ENCRYPT_STRUCTURES files that are provided with the kit.

The examples directory, ENCRYPT$EXAMPLES, has a copy of the ENCRYPT_STRUCTURES file in each supported programming language.

4.5.3 Error Handling

By default, Encryption signals error conditions with messages. To intercept a message that is inappropriate for your application, supply a condition handler.

For information about implementing condition handlers, see your programming language reference manual. For additional documentation, see:

- The *OpenVMS Programming Interfaces: Calling a System Routine* — provides full information about data types, access type, and passing mechanisms
- ENCRYPT$EXAMPLES:ENCRYPT$EXAMPLES.TXT — online file that contains descriptions of files with sample application programs

ENCRYPT$DECRYPT

Decrypts the next record of ciphertext according to the algorithm specified in the ENCRYPT$INIT call.

Format

ENCRYPT$DECRYPT   context ,input ,output [,output-length] [,p1]

Arguments

context

type:	longword integer (signed)
access:	write only
mechanism:	by reference

Context area initialized when ENCRYPT$INIT completes execution. The context argument is the address of a longword of unspecified interpretation that is used to convey context between encryption operations.

input

type:	char_string
access:	read only
mechanism:	by descriptor

Ciphertext record that ENCRYPT$DECRYPT is to decrypt. The input argument is the address of a descriptor pointing to a byte-aligned buffer containing the input record to the decryption operation.

output

type:	char_string
access:	write only
mechanism:	by descriptor

Plaintext record that results when ENCRYPT$DECRYPT completes execution. The output argument is the address of a descriptor pointing to a byte-aligned buffer that will contain the output record from the decryption operation.

If the descriptor is dynamic and insufficient space is allocated to contain the output record, storage will be allocated from dynamic memory. If insufficient space exists to contain the output of the operation, then an error status is returned.

The ENCRYPT$DECRYPT routine adjusts the length of the output descriptor, if possible, to reflect the actual length of the output string. If the descriptor type is not DSC$K_DTYPE_VS (varying string), DSC$K_DTYPE_V (varying), or DSC$K_DTYPE_D (dynamic), the routine takes the actual output count from the **output-length** argument.

output-length

type:	word integer
access:	write only
mechanism:	by reference

Optional argument.

Number of bytes that ENCRYPT$DECRYPT wrote to the output buffer. The **output-length** argument is the address of a word containing the number of bytes written to the output buffer, including any bytes of pad characters generated by the selected algorithm to meet length requirements of the input buffer, if any. Output length does not count padding in the case of a fixed-length string.

Some encryption algorithms have specific requirements for the length of the input and output strings. In particular, DESECB and DESCBC pad input data with from 1 to 7 bytes to form complete 64-bit blocks for operation. The values of the pad characters are indeterminate.

When you decrypt fewer than 8 bytes, present the full 8 bytes resulting from the ENCRYPT$ENCRYPT to ENCRYPT$DECRYPT. Retain the byte count of the input data in order to strip trailing pad bytes after a subsequent decryption operation.

p1

type:	quadword
access:	read only
mechanism:	by reference

Optional argument. The p1 argument is the address of a quadword initialization vector used to seed the two modes of the DES algorithm for which it is applicable (DESECB and DESCFB). If this argument is omitted, the initialization vector used is the residue of the previous use of the specified context block. ENCRYPT$INIT initializes the context block with an initialization vector of zero.

Description

The ENCRYPT$DECRYPT routine decrypts the next record of ciphertext according to the algorithm specified in the ENCRYPT$INIT call. Any errors encountered in the operation are returned as status values. The message authentication mode (DESMAC) is not supported by ENCRYPT$DECRYPT.

The ENCRYPT$DECRYPT routine returns a 32-bit status code indicating the success or failure of the routine's operation.

Condition Values Returned

SS$_NORMAL	Record successfully decrypted.
ENCRYPT$*xyz*	An error reported by the Encryption software. *xyz* identifies the message (see Appendix B).
SS$_*xyz*	A return status from a called system service. *xyz* identifies the return status.

ENCRYPT$DECRYPT_ONE_RECORD

Decrypts a small amount of data on a decrypt stream.

Format

ENCRYPT$DECRYPT_ONE_RECORD   input ,output ,key-name ,algorithm

Arguments

input

type:	char_string
access:	read only
mechanism:	by descriptor

Ciphertext record to be decrypted. The **input** argument is the address of a string descriptor pointing to a byte-aligned buffer containing the input record to be decrypted.

output

type:	char_string
access:	write only
mechanism:	by descriptor

Plaintext record resulting when ENCRYPT$DECRYPT_ONE_RECORD completes execution. The **output** argument is the address of a string descriptor pointing to a byte-aligned buffer that will contain the plaintext record.

If the descriptor is dynamic and insufficient space is allocated to contain the output record, storage is allocated from dynamic memory. If insufficient space exists to contain the output of the operation, an error is returned.

The ENCRYPT$DECRYPT_ONE_RECORD routine adjusts the length of the output descriptor, if possible, to reflect the actual length of the output string.

key-name

type:	char_string
access:	read only
mechanism:	by descriptor

Key used to initialize the decrypt stream. The **key-name** argument is the address of a string descriptor pointing to the name of the previously defined user key to be used.

algorithm

type:	char_string
access:	read only
mechanism:	by descriptor

Algorithm used for the decryption operation. The **algorithm** argument is the address of a string descriptor pointing to a code for the selected algorithm. The algorithm code is an ASCII string. Specify the descriptor type value as one of the following:

- DSC$K_DTYPE_T (text)
- DSC$K_DTYPE_VT (varying text)

- DSC$K_DTYPE_Z (unspecified)

The following algorithms are valid:

- DESCBC (default)
- DESECB
- DESCFB

Description

In some applications, only a small amount of data needs to be decrypted on a particular decrypt stream. The ENCRYPT$DECRYPT_ONE_RECORD routine allows you to perform such a decryption operation.

The ENCRYPT$DECRYPT_ONE_RECORD routine is a shorthand form of the ENCRYPT$INIT, ENCRYPT$DECRYPT, and ENCRYPT$FINI sequence of calls. However, using ENCRYPT$DECRYPT_ONE_RECORD repeatedly to decrypt records of a file is extremely inefficient.

The ENCRYPT$DECRYPT_ONE_RECORD routine returns a 32-bit status code indicating the success or failure of the routine's operation.

Condition Values Returned

SS$_NORMAL	Operation performed.
ENCRYPT$*xyz*	An error reported by the Encryption software. *xyz* identifies the message (see Appendix B).
SS$_*xyz*	A return status from a called system service. *xyz* identifies the return status.

ENCRYPT$DEFINE_KEY

Places a key definition into the process, group, job, or system key storage table.

Format

ENCRYPT$DEFINE_KEY key-name ,key-value ,key-flags

Arguments

key-name

type:	char_string
access:	read only
mechanism:	by descriptor

Name of the key defined when ENCRYPT$DEFINE_KEY completes execution. The **key-name** argument is the address of a string descriptor pointing to a char_string that is interpreted as the name of the key to be defined. A maximum of 243 characters is permitted.

Note

Key names beginning with ENCRYPT$ are reserved for Compaq.

key-value

type:	char_string
access:	read only
mechanism:	by descriptor

Key value defined when ENCRYPT$DEFINE_KEY completes execution. The *key-value* argument is the address of a string descriptor pointing to a vector of unsigned byte values that are assigned to the named key. A maximum of 240 bytes may be assigned.

key-flags

type:	longword
access:	read only
mechanism:	by reference

Flags that ENCRYPT$DEFINE_KEY uses when defining a key. The **key-flags** argument is the address of a longword containing flags that control the key definition process.

Each flag has a symbolic name. The constants associated with these names are defined in the ENCRYPT$EXAMPLES:ENCRYPT_STRUCTURES files in various programming languages. Table 4–3 defines the function of each flag.

Table 4–3 ENCRYPT$DEFINE_KEY Flags

Flag Symbolic Name	Function Function
ENCRYPT$M_KEY_PROCESS	Places definition in process table
ENCRYPT$M_KEY_GROUP	Places definition in group table
ENCRYPT$M_KEY_JOB	Places definition in job table
ENCRYPT$M_KEY_SYSTEM	Places definition in system table
ENCRYPT$M_KEY_LITERAL	Stores key without compressing

Description

The ENCRYPT$DEFINE_KEY routine places a key definition into the process, group, job, or system key storage table. The key value supplied with the routine is processed as specified and placed in the key storage table under the indicated name. The ENCRYPT$DEFINE_KEY routine does not interpret the key value.

By default, keys are treated as char_string keys, using the Digital Multinational Character Set and are compressed before being inserted into the key storage table. The compression proceeds as follows:

1. The string is converted to uppercase characters.
2. The digits 0 through 9 are left unchanged.
3. All characters except letters, digits, dollar signs, periods, and underscores are converted to spaces.
4. All sequences of multiple spaces (or characters that have been converted into spaces) are converted into single spaces.

When a char_string key is retrieved from key storage for use as a DES key, it is folded into an 8-byte key by exclusive OR-ing 8-byte segments of the key string together, and then applying odd parity to each byte by modifying the sign bit (bit 7).

The key flag ENCRYPT$M_KEY_LITERAL specifies that the key string supplied is a binary key. A binary key is not compressed, but is placed into key storage as is. When a binary key is used as a DES key, it is likewise folded into an 8-byte key by exclusive OR-ing 8-byte segments together. Odd parity is then applied by modifying the low bit (bit 0) of each byte.

The ENCRYPT$DEFINE_KEY routine returns a 32-bit status code indicating the success or failure of the routine's operation.

Condition Values Returned

SS$_NORMAL	Key has been defined.
ENCRYPT$*xyz*	An error reported by the Encryption software. *xyz* identifies the message (see Appendix B).
SS$_*xyz*	A return status from a called system service. *xyz* identifies the return status.

ENCRYPT$DELETE_KEY

Deletes a key definition from a key storage table.

Format

ENCRYPT$DELETE_KEY key-name ,key-flags

Arguments

key-name

type:	char_string
access:	read only
mechanism:	by descriptor

Name of the key removed from a key storage table when ENCRYPT$DELETE_KEY completes execution. The **key-name** argument is the address of a string descriptor pointing to a char_string that is interpreted as the name of the key to be deleted. A maximum of 243 characters is permitted.

key-flags

type:	longword
access:	read only
mechanism:	by reference

Key table from which ENCRYPT$DELETE_KEY removes a key. The key-flags argument is a longword containing flags that control the deletion process. The following flags are available:

ENCRYPT$M_KEY_PROCESS	Deletes a key from process table
ENCRYPT$M_KEY_GROUP	Deletes a key from group table
ENCRYPT$M_KEY_JOB	Deletes a key from job table
ENCRYPT$M_KEY_SYSTEM	Deletes a key from system table

Description

The ENCRYPT$DELETE_KEY routine deletes a key definition from a key storage table. The ENCRYPT$DELETE_KEY routine returns a 32-bit status code indicating the success or failure of the routine's operation.

Condition Values Returned

SS$_NORMAL	Key has been deleted.
ENCRYPT$*xyz*	An error reported by the Encryption software. *xyz* identifies the message (see Appendix B).
SS$_*xyz*	A return status from a called system service. *xyz* identifies the return status.

ENCRYPT$ENCRYPT

Transforms the next record of plaintext according to the algorithm you specify in the ENCRYPT$INIT call. This routine performs either an encryption or decryption operation.

Format

ENCRYPT$ENCRYPT context ,input ,output [,output-length] [,p1]

Arguments

context

type:	longword integer (signed)
access:	write only
mechanism:	by reference

Context area initialized when ENCRYPT$INIT completes execution. The **context** argument is the address of a longword of unspecified interpretation that is used to convey context between encryption operations.

input

type:	char_string
access:	read only
mechanism:	by descriptor

Plaintext record to encrypt. The **input** argument is the address of a descriptor pointing to a byte-aligned buffer containing the input record to the encryption operation.

output

type:	char_string
access:	write only by descriptor
mechanism:	

Ciphertext record that results when ENCRYPT$ENCRYPT completes execution. The **output** argument is the address of a descriptor pointing to a byte-aligned buffer that will contain the output record from the encryption operation.

If the descriptor is dynamic and insufficient space is allocated to contain the output record, storage is allocated from dynamic memory.

ENCRYPT$ENCRYPT adjusts the length of the output descriptor, if possible, to reflect the actual length of the output string. If the descriptor type is not DSC$K_DTYPE_VS (varying string), DSC$K_DTYPE_V (varying), or DSC$K_DTYPE_D (dynamic), the routine takes the actual output count from the **output-length** argument.

output-length

type:	word integer
access:	write only
mechanism:	by reference

Optional argument. Number of bytes that ENCRYPT$ENCRYPT wrote to the output buffer. The **output-length** argument is the address of a word containing the number of bytes written to the output buffer.

Some encryption algorithms have specific requirements for the length of the input and output strings. In particular, DESECB and DESCBC pad input data with from 1 to 7 bytes to form complete 64-bit blocks for operation. The values of the pad characters are indeterminate.

When you decrypt fewer than 8 bytes, preserve and present to ENCRYPT$DECRYPT the full 8 bytes resulting from ENCRYPT$ENCRYPT. Retain the byte count of the input data in order to strip trailing pad bytes after a subsequent decryption operation.

p1

type:	quadword
access:	read only
mechanism:	by reference

Optional argument. The **p1** argument is the address of a quadword initialization vector used to seed the three modes (DESECB, DESCFB, and DESMAC) of the DES algorithm for which it is applicable.

If you omit this argument, the initialization vector used is the residue of the previous use of the specified context block. ENCRYPT$INIT initializes the context block with an initialization vector of zero.

Description

The ENCRYPT$ENCRYPT routine transforms the next record of plaintext according to the algorithm specified in the ENCRYPT$INIT call. Any errors encountered in the operation are returned as status values. The ENCRYPT$ENCRYPT routine returns a 32-bit status code indicating the success or failure of the routine's operation.

Condition Values Returned

SS$_NORMAL	Record successfully encrypted.
ENCRYPT$*xyz*	An error reported by the Encryption software. *xyz* identifies the message (see Appendix B).
SS$_*xyz*	A return status from a called system service. *xyz* identifies the return status.

ENCRYPT$ENCRYPT_FILE

Encrypts or decrypts data files.

Format

ENCRYPT$ENCRYPT_FILE input-file ,output-file ,key-name ,algorithm ,file-flags ,item-list

Arguments

input-file

type:	char_string
access:	read only
mechanism:	by descriptor

Name of the input file that ENCRYPT$ENCRYPT_FILE is to process. The **input-file** argument is the address of a string descriptor pointing to the file specification string for the input file.

Wildcard characters are valid. To specify multiple input files, you must use wildcard characters.

output-file

type:	char_string
access:	read only
mechanism:	by descriptor

Name of the output file that ENCRYPT$ENCRYPT_FILE is to generate. The **output-file** argument is the address of a string descriptor pointing to the file specification for the output file to be processed.

You can use wildcard characters. To specify the same names for the output and input files, use a null character as the **output-file** argument.

key-name

type:	char_string
access:	read only
mechanism:	by descriptor

Name of the key used when ENCRYPT$ENCRYPT_FILE processes files. The **key-name** argument is the address of a string descriptor pointing to the name of the key to be used in initializing the encrypt or decrypt stream used for each file processed.

algorithm

type:	char_string
access:	read only
mechanism:	by descriptor

Name of the algorithm that ENCRYPT$ENCRYPT_FILE uses to initialize the process stream. The **algorithm** argument is the address of a string descriptor pointing to the name of the algorithm.

The following algorithms are valid:

- DESCBC (default)
- DESECB

- DESCFB

file-flags

type:	longword
access:	read only
mechanism:	by reference

Flags that specify how ENCRYPT$ENCRYPT_FILE performs the file operation. The **file-flags** argument is the address of a longword containing a mask of flags. Table 4–4 shows the function of each flag.

Table 4–4 ENCRYPT$ENCRYPT_FILE Flags

Flag	Function
ENCRYPT$M_FILE_COMPRESS	Compresses file data before encryption.
ENCRYPT$M_FILE_ENCRYPT	Flag set: encrypts the file. Flag clear: decrypts the file.
ENCRYPT$M_FILE_DELETE	Deletes the input file when the operation completes.
ENCRYPT$M_FILE_ERASE	Erases the file with the security data pattern before deleting it.
ENCRYPT$M_FILE_KEY_VALUE	Flag set: Treats the key value as a literal value and does not compress it. Flag clear: Treats the key value as a text string that can be compressed. If the KEY_NAME parameter is present, this flag is ignored.

item-list

type:	item_list_3
access:	read only
mechanism:	by descriptor

Item list used to specify additional arguments for the ENCRYPT$ENCRYPT_FILE routine. The **item-list** argument is the address of an item list. The following item-code is valid:

ENCRYPT$K_DATA_ALGORITHM

type:	3 longwords
access:	read only
mechanism:	by descriptor

Algorithm to be used to encrypt the file. This argument specifies the address and length of the name string of the algorithm.

The following algorithms are valid:

- DESCBC (default)
- DESECB
- DESCFB

Description

The ENCRYPT$ENCRYPT_FILE routine either encrypts or decrypts data files from within an application.

The routine uses the user key and the specified algorithm to protect only the randomly generated key and the initialization vector that are used with the DESCBC algorithm to encrypt the file.

The ENCRYPT$ENCRYPT_FILE routine returns a 32-bit status code indicating the success or failure of the routine's operation.

When you use this routine, do not also use ENCRYPT$INIT or ENCRYPT$FINI.

Condition Values Returned

SS$_NORMAL	Record successfully encrypted.
ENCRYPT$*xyz*	An error reported by the Encryption software. *xyz* identifies the message (see Appendix B).
SS$_*xyz*	A return status from a called system service. *xyz* identifies the return status.

ENCRYPT$ENCRYPT_ONE_RECORD

Encrypts a small amount of data in an encrypt stream.

Format

ENCRYPT$ENCRYPT_ONE_RECORD input ,output ,key-name ,algorithm

Arguments

input

type:	char_string
access:	read only
mechanism:	by descriptor

Plaintext record to be encrypted. The **input** argument is the address of a string descriptor pointing to a byte-aligned buffer containing the input record to be encrypted.

output

type:	char_string
access:	write only
mechanism:	by descriptor

Ciphertext record resulting when the routine completes execution. The **output** argument is the address of a string descriptor pointing to a byte-aligned buffer that will contain the ciphertext record.

If the descriptor is dynamic, and insufficient space is allocated to contain the output record, storage is allocated from dynamic memory. If insufficient space exists to contain the output of the operation, an error is returned.

The ENCRYPT$ENCRYPT_ONE_RECORD routine adjusts the length of the output descriptor, if possible, to reflect the actual length of the output string.

key-name

type:	char_string
access:	read only
mechanism:	by descriptor

Key used to initialize the encrypt stream. The **key-name** argument is the address of a string descriptor pointing to the name of the previously defined user key to be used.

algorithm

type:	char_string
access:	read only
mechanism:	by descriptor

Algorithm used for the encryption operation. The **algorithm** argument is the address of a string descriptor pointing to a code for the selected algorithm. The algorithm code is an ASCII string. For descriptor type value, use one of the following:

- DSC$K_DTYPE_T (text)
- DSC$K_DTYPE_VT (varying text)

- DSC$K_DTYPE_Z (unspecified)

The following algorithms are valid:

- DESCBC (default)
- DESECB
- DESCFB

Description

To encrypt only a small amount of data, use the ENCRYPT$ENCRYPT_ONE_RECORD routine.

The ENCRYPT$ENCRYPT_ONE_RECORD routine is a shorthand form of the ENCRYPT$INIT, ENCRYPT$ENCRYPT, and ENCRYPT$FINI sequence of calls. However, using ENCRYPT$ENCRYPT_ONE_RECORD repeatedly to encrypt records of a file is extremely inefficient.

The ENCRYPT$ENCRYPT_ONE_RECORD routine returns a 32-bit status code indicating the success or failure of the routine's operation.

Condition Values Returned

SS$_NORMAL	Operation performed.
ENCRYPT$*xyz*	An error reported by the Encryption software. *xyz* identifies the message (see Appendix B).
SS$_*xyz*	A return status from a called system service. *xyz* identifies the return status.

ENCRYPT$FINI

Disassociates the encryption context and releases it.

Format

ENCRYPT$FINI context

Arguments

context

type:	longword integer (signed)
access:	read/write
mechanism:	by reference

Context area terminated when ENCRYPT$FINI completes execution. The **context** argument is the address of a longword initialized by the ENCRYPT$INIT routine.

Description

The ENCRYPT$FINI routine disassociates the indicated encryption context and releases it. The ENCRYPT$FINI routine returns a 32-bit status code indicating the success or failure of the routine's operation.

Condition Values Returned

SS$_NORMAL	Encryption context successfully terminated.
ENCRYPT$*xyz*	An error reported by the Encryption software. *xyz* identifies the message (see Appendix B).
SS$_*xyz*	A return status from a called system service. *xyz* identifies the return status.

ENCRYPT$GENERATE_KEY

Generates a random key value.

Format

```
ENCRYPT$GENERATE_KEY   algorithm-name ,key-length [,factor-a] [,factor-b]
                       [,factor-c] [,key buffer]
```

Arguments

algorithm-name

type:	char_string
access:	read only
mechanism:	by descriptor

The name of the algorithm that will use the generated key.

key-length

type:	word unsigned
access:	read only
mechanism:	by reference

Unsigned integer indicating the size of the key to be generated. The **key-length** argument is the address of an unsigned word containing a value that indicates the length of the key.

factor-a, factor-b, factor-c

type:	char_string
access:	read only
mechanism:	by descriptor

Optional arguments. The **factor-a**, **factor-b**, and **factor-c** arguments are operation-dependent data used as randomizing factors when the routine generates a key value. For example, the factors might be:

- Time an operation started
- Size of a certain stack
- Copy of the last command line

key-buffer

type:	char_string
access:	write
mechanism:	by descriptor

Buffer into which the generated key is to be placed. The **key-buffer** argument is the address of a string descriptor referencing the appropriate buffer.

If you specify a class D descriptor, dynamic memory is allocated to contain the entire key.

Description

The ENCRYPT$GENERATE_KEY routine generates a random key value. The ENCRYPT$GENERATE_KEY routine returns a 32-bit status code indicating the success or failure of the routine's operation.

Condition Values Returned

SS$_NORMAL	Key has been created.
ENCRYPT$*xyz*	An error reported by the Encryption software. *xyz* identifies the message (see Appendix B).
SS$_*xyz*	A return status from a called system service. *xyz* identifies the return status.

ENCRYPT$INIT

Initializes the context for the encryption operation.

Format

ENCRYPT$INIT   context ,algorithm ,key-type ,key-name [,p1]

Arguments

context

type:	longword integer signed
access:	write only
mechanism:	by reference

Context area that is initialized. The **context** argument is the address of a longword of unspecified interpretation that is used to convey context between encryption operations. An uninitialized context longword is defined to be zero and is initialized to nonzero by this routine. The context area itself is allocated from process dynamic memory.

algorithm

type:	char_string
access:	read/write
mechanism:	by descriptor

Algorithm used for the encryption operation. The **algorithm** argument is the address of a string descriptor pointing to a code for the selected algorithm. The algorithm code is an ASCII string. For descriptor type value, use one of the following:

DSC$K_DTYPE_T (text)
DSC$K_DTYPE_VT (varying text)
DSC$K_DTYPE_Z (unspecified)

The following algorithms are valid:

- DESCBC (default)
- DESECB
- DESCFB

key-type

type:	longword logical unsigned
access:	read only
mechanism:	by reference

Code specifying how ENCRYPT$INIT is to interpret the **key-name** argument. The **key-type** argument is the address of an unsigned longword indicating whether key-name is the name of the key or the key value. If you specify:

Key-type as 0	ENCRYPT$INIT interprets **key-name** as a descriptor pointing to the key name string.
Key-type as 1	ENCRYPT$INIT interprets **key-name** as the descriptor for the value of the key to be used.

key-name

type:	char_string
access:	read only
mechanism:	by descriptor

Key that ENCRYPT$INIT passes to the selected encryption routine. The **key-name** argument is the address of a character string descriptor containing the name of the key or the address of the actual key value. ENCRYPT$INIT's interpretation of this argument depends on the value of key-type. If this argument is:

The key name	Actual key value is retrieved from key storage by the selected encryption routine.
A key value	It is stored with a temporary name, which is passed to the selected encryption routine.

If the **key-name** argument is used to specify a key value (that is, if key-type has been specified as 1), the key-name string descriptor type field determines whether the key value is to be treated as a char_string or as a binary value to be used exactly as specified.

If the descriptor type is DSC$K_DTYPE_T (char_string), DSC$K_DTYPE_VT (varying char_string), or DSC$K_DTYPE_Z (unspecified), the value is treated as a text string to be compressed.

All other descriptor types are treated as though the key value is to be used exactly as specified.

p1

type:	quadword
access:	read only
mechanism:	by reference

Optional argument. The **p1** argument is the address of a quadword initialization vector used to seed the three modes of the DES algorithm that uses an initialization vector. These modes are: DESCBC (default), DESCFB, and DESMAC.

Description

ENCRYPT$INIT initializes the context for the encryption operation. ENCRYPT$INIT creates pre-initialized key tables in the context area to speed the encryption or decryption process. Before you can re-use a context with a new algorithm, key, or other values specified with ENCRYPT$INIT, terminate the old context with a call to ENCRYPT$FINI.

Note

Always initialize the context with ENCRYPT$INIT when you change the operation from encryption to decryption, or from decryption to encryption.

ENCRYPT$INIT returns a 32-bit status code indicating the success or failure of the routine's operation.

Condition Values Returned

SS$_NORMAL	Initialization successfully completed.
ENCRYPT$*xyz*	An error reported by the Encryption software. *xyz* identifies the message (see Appendix B).
SS$_*xyz*	A return status from a called system service. *xyz* identifies the return status.

ENCRYPT$STATISTICS

Gains access to the statistics maintained by the Encryption software.

Format

ENCRYPT$STATISTICS   context ,code ,destination ,return-length

Arguments

context

type:	longword
access:	read only
mechanism:	by reference

Context area initialized by ENCRYPT$INIT. The **context** argument is the address of a longword initialized by the ENCRYPT$INIT routine.

code

type:	longword
access:	read only
mechanism:	by reference

Code specifying the desired statistic. The **code** argument is the address of a longword containing the code. The only accepted value is 1, which indicates that ENCRYPT$STATISTICS is to return all statistics to the destination buffer.

destination

type:	char_string
access:	write only
mechanism:	by descriptor

Buffer into which ENCRYPT$STATISTICS places the statistics. The **destination** argument is the address of a string descriptor describing the buffer. Ensure that the destination buffer is at least 20 bytes long and contains:

- One longword indicating the number of times the primitive has been entered referencing this encryption stream
- One quadword indicating the total bytes processed for this stream
- One quadword indicating the total CPU time, in OpenVMS time format, spent on processing requests for this stream

return-length

type:	longword
access:	write only
mechanism:	by reference

Number of bytes written to the destination buffer. The **return-length** argument is the address of a word containing the number of bytes.

Description

To track the progress and performance of an encryption operation, the Encryption for OpenVMS software maintains statistics in the context area. You can access these statistics with the ENCRYPT$STATISTICS routine. The ENCRYPT$STATISTICS routine returns a 32-bit status code indicating the success or failure of the routine's operation.

Condition Values Returned

SS$_NORMAL	Statistics returned.
ENCRYPT$*xyz*	An error reported by the Encryption software. *xyz* identifies the message (see Appendix B).
SS$_*xyz*	A return status from a called system service. *xyz* identifies the return status.

A
Command Reference

This appendix describes the syntax of the Encryption for OpenVMS commands and qualifiers. The command descriptions are presented in alphabetical order.

The Encryption-related commands are:

- BACKUP /ENCRYPT
- DECRYPT
- ENCRYPT
- ENCRYPT /AUTHENTICATE
- ENCRYPT /CREATE_KEY
- ENCRYPT /REMOVE_KEY

Enter these commands at the DCL prompt ($).

BACKUP /ENCRYPT

Creates and restores encrypted save sets. Specify the /ENCRYPT qualifier in any part of the BACKUP command line.

For additional security, specify the /ENCRYPT qualifier with no parameters and press Return. The command prompts you for a key value. When you enter a value, the software does not echo what you type and, for verification, prompts you to retype the value.

Format

BACKUP /ENCRYPT=(NAME=*key-name* | VALUE=*key-value* [, ALGORITHM=*algorithm*])

/ENCRYPT Values

NAME=*key-name*
Required if you do not specify *key-value*.

Existing key name previously created and stored in the key storage table with the ENCRYPT /CREATE_KEY command.

Specify either the name or the value of a key, but not both. For added security, specify neither, and the system prompts you. Your input is not echoed.

VALUE=*key-value*
Required if you do not specify *key-name*.

Interactively defines a value for the key. Specify one of the following:

- Character string enclosed in quotation marks (" "). Specify from 1 to 243 alphanumeric characters. Dollar signs and underscores are valid.
- Hexadecimal constant using the digits 0 to 9 and A to F.

Specify either the name or the value of a key, but not both. For added security, specify neither, and the system prompts you. Your input is not echoed.

ALGORITHM=DESCBC (default) | DESECB | DESCFB
Algorithm used to encrypt the initialization vector and the key you supply.

DECRYPT

Decrypts files previously encrypted with the ENCRYPT command.

Format

DECRYPT *input-file key-name* [*qualifiers*]

Parameters

input-file
File names of the files to decrypt. If you use wildcard characters, do not include directory files or files with bad blocks.

key-name
Key name previously stored in the key storage table with the ENCRYPT /CREATE_KEY command.

Qualifiers

/BACKUP[=*time*]
Selects files according to the dates of their most recent backup.

This qualifier is relevant only when used with the /BEFORE or the /SINCE qualifier. In addition, do not use /BACKUP with /EXPIRED or /MODIFIED.

If you omit *time*, TODAY is used. For more information on time specifications, see the *OpenVMS User's Manual*.

/BEFORE[=*time*]
Selects files that have a creation time before the time you specify.

If you omit *time*, TODAY is used. For more information on time specifications, see the *OpenVMS User's Manual*.

/BY_OWNER[=*uic*]
/NOBY_OWNER
Selects files with the owner UIC you specify.

If you omit *uic*, the UIC of the current process is used. For more information on specifying UIC format, see the *OpenVMS User's Manual*.

/CONFIRM
/NOCONFIRM
Controls whether or not a confirmation request is displayed before each decryption, as follows:

Response	Meaning
YES	Decrypts the file
NO or Return	Does not decrypt the file (default)
QUIT or Ctrl/Z	Does not decrypt the file or any subsequent files
ALL	Decrypts the file plus all subsequent files

/DELETE
/NODELETE
Default: /NODELETE.

Controls whether or not the input files are deleted after the decryption operation is complete and the output file is written and closed.

/ERASE
/NOERASE
Controls whether or not the input files are erased with the data security pattern before being deleted. By default, the location in which the data was stored is not overwritten with the data security pattern. The /ERASE qualifier must be used with /DELETE.

/EXCLUDE=*file-spec*
/NOEXCLUDE
Excludes the specified files from the decryption operation. You can use wildcard characters. You do not need to enter an entire file specification. Any field that you omit defaults to the input file specification.

Because directory files are never encrypted, you need not specify them.

/EXPIRED[=*time*]
Selects files according to the dates on which they expire.

This qualifier is relevant only when used with the /BEFORE or the /SINCE qualifier. In addition, do not use /EXPIRED with /BACKUP or /MODIFIED.

If you omit *time*, TODAY is used. For more information on time specifications, see the *OpenVMS User's Manual*.

/KEY_ALGORITHM=DESCBC (default) | DESECB | DESCFB
Algorithm by which the key and the initialization vector are protected within the encrypted file. Specify the same algorithm that you used to encrypt the key, if not the default.

/MODIFIED[=*time*]
Selects files according to the dates on which they were last modified.

This qualifier is relevant only when used with the /BEFORE or the /SINCE qualifier. In addition, do not use /MODIFIED with /BACKUP or /EXPIRED.

If you omit *time*, TODAY is used. For more information on time specifications, see the *OpenVMS User's Manual*.

/OUTPUT=*file-spec*
Alternate output file name for the decryption operation.

By default, each input file decrypted is written to a separate output file that is one version higher than that of the input file. When using the /OUTPUT qualifier, specify the parts of the file specification different from the defaults. You do not need to provide an entire file specification. Any field that you omit defaults to the input file specification.

/SHOW=(*keyword-list*)
Controls whether or not the following information about the decryption operation is displayed on SYS$COMMAND:

Keyword	Meaning
FILES	Displays input and output file names on SYS$COMMAND
STATISTICS	Displays the encryption stream statistics: • Bytes processed • Internal records processed • CPU time consumed within the encryption algorithm

/SINCE[=*time*]

Selects files that have a creation date before the time you specify.

If you omit *time*, TODAY is used. For more information on time specifications, see the *OpenVMS User's Manual*.

/STATISTICS

Same as /SHOW=STATISTICS.

ENCRYPT

Encrypts files. Before you enter this command, create a key with the ENCRYPT /CREATE_KEY command.

Format

ENCRYPT *input-file key-name* [*qualifiers*]

Parameters

input-file
File names of the files to encrypt. If you use wildcard characters, do not include directory files or files with bad blocks.

key-name
Key name previously stored in the key storage table with the ENCRYPT /CREATE_KEY command.

Qualifiers

/BACKUP[=*time*]
Selects files according to the dates of their most recent backup.

This qualifier is relevant only when used with the /BEFORE or the /SINCE qualifier. In addition, do not use /BACKUP with /EXPIRED or /MODIFIED.

If you omit *time*, TODAY is used. For more information on time specifications, see the *OpenVMS User's Manual*.

/BEFORE[=*time*]
Selects files that have a creation time before the time you specify.

If you omit *time*, TODAY is used. For more information on time specifications, see the *OpenVMS User's Manual*.

/BY_OWNER[=*uic*]
/NOBY_OWNER
Selects files with the owner UIC you specify.

If you omit *uic*, the UIC of the current process is used. For more information on specifying UIC format, see the *OpenVMS User's Manual*.

/COMPRESS
/NOCOMPRESS
Optional. Default: /NOCOMPRESS.

Controls whether or not data compression occurs before a file is encrypted.

/CONFIRM
/NOCONFIRM
Controls whether or not a confirmation request is displayed before each encryption, as follows:

Response	Meaning
YES	Encrypts the file
NO or Return	Does not encrypt the file (default)
QUIT or Ctrl/Z	Does not encrypt the file or any subsequent files
ALL	Encrypts the file plus all subsequent files

/DATA_ALGORITHM=DESCBC (default) | DESECB | DESCFB
Algorithm used to encrypt the file. It is used with the randomly generated key to perform encryption.

/DELETE
/NODELETE
Controls whether or not the input files are deleted after the encryption operation is complete and the output file is written and closed. By default, the input file is not deleted.

/ERASE
/NOERASE
Controls whether or not the input files are erased with the data security pattern before being deleted. By default, the location in which the data was stored is not overwritten with the data security pattern. The /ERASE qualifier must be used with /DELETE.

/EXCLUDE=*file-spec*
/NOEXCLUDE
Excludes the specified files from the encryption operation. You can use wildcard characters. You do not need to enter an entire file specification. Any field that you omit defaults to the input file specification.

Beacuse directory files are never encrypted, you need not specify them.

/EXPIRED[=*time*]
Selects files according to the dates on which they expire.

This qualifier is relevant only when used with the /BEFORE or the /SINCE qualifier. In addition, do not use /EXPIRED with /BACKUP or /MODIFIED.

If you omit *time*, TODAY is used. For more information on time specifications, see the *OpenVMS User's Manual*.

/KEY_ALGORITHM=DESCBC (default) | DESECB | DESCFB
Algorithm by which the key and the initialization vector are to be protected within the encrypted file. The command uses this algorithm with the key you supply to encrypt the randomly generated data encryption key and the initialization vector stored within the file.

/MODIFIED[=*time*]
Selects files according to the dates on which they were last modified.

This qualifier is relevant only when used with the /BEFORE or the /SINCE qualifier. In addition, do not use /MODIFIED with /BACKUP or /EXPIRED.

If you omit *time*, TODAY is used. For more information on time specifications, see the *OpenVMS User's Manual*.

/OUTPUT=*file-spec*

Alternate output file name for the encryption operation. By default, each input file encrypted is written to a separate output file that is one version higher than the highest version of the input file. When using the /OUTPUT qualifier, specify the parts of the file specification different from the defaults. You do not need to provide an entire file specification. Any field that you omit defaults to the input file specification.

/SHOW=*keyword-list*

Controls whether or not the following information about the encryption operation is displayed on SYS$COMMAND:

Keyword	Meaning
FILES	Displays input and output file names on SYS$COMMAND
STATISTICS	Displays the encryption stream statistics: • Bytes processed • Internal records processed • CPU time consumed within the encryption algorithm

/SINCE[=*time*]

Selects files that have a creation date before the time you specify.

If you omit *time*, TODAY is used. For more information on time specifications, see the *OpenVMS User's Manual*.

/STATISTICS

Same as /SHOW=STATISTICS.

/VERSION

Displays the version number of the Encryption for OpenVMS software running on your system.

ENCRYPT /AUTHENTICATE

Associates a Message Authenticate Code (MAC) value with one or more files and checks for any modification of either plaintext or ciphertext files.

Format

ENCRYPT /AUTHENTICATE *file-spec key-name* [*qualifiers*]

Parameters

file-spec
File names of the files to authenticate. Behavior can be modified with the /MULTIPLE_FILES qualifier.

key-name
Key name previously stored in the key storage table with the ENCRYPT /CREATE_KEY command.

Qualifiers

/BACKUP[=*time*]
Selects files according to the dates of their most recent backup.

This qualifier is relevant only when used with the /BEFORE or the /SINCE qualifier. In addition, do not use /BACKUP with /EXPIRED or /MODIFIED.

If you omit *time*, TODAY is used. For more information on time specifications, see the *OpenVMS User's Manual*.

/BEFORE=*time*
Selects files that have a creation time before the time you specify.

If you omit *time*, TODAY is used. For more information on time specifications, see the *OpenVMS User's Manual*.

/BY_OWNER[=*uic*]
/NOBY_OWNER
Selects files with the owner UIC you specify.

If you omit *uic*, the UIC of the current process is used. For more information on specifying UIC format, see the *OpenVMS User's Manual*.

/CONFIRM
/NOCONFIRM
Controls whether or not a confirmation request is displayed before each authentication, as follows:

Response	Meaning
YES	Authenticates the file
NO or Return	Does not authenticate the file (default)
QUIT or Ctrl/Z	Does not authenticate the file or any subsequent files

Response	Meaning
ALL	Encrypts the file plus all subsequent files

/DATABASE=*file-spec*
/NODATABASE
File name of the file in which to store binary MAC values.

Generates a MAC using the file contents. If you do not specify a file name, the file name SYS$LOGIN:ENCRYPT$MAC.DAT is used.

/EXCLUDE=*file-spec*
/NOEXCLUDE
Excludes the specified files from the authentication operation. You can use wildcard characters. You do not need to enter an entire file specification. Any field that you omit defaults to the input file specification.

Because directory files are never encrypted, you need not specify them.

/EXPIRED[=*time*]
Selects files according to the dates on which they expire.

This qualifier is relevant only when used with the /BEFORE or the /SINCE qualifier. In addition, do not use /EXPIRED with /BACKUP or /MODIFIED.

If you omit a time value, TODAY is used. For more information on time specifications, see the *OpenVMS User's Manual*.

/LOG
Displays the results of the authentication operation.

/MODIFIED[=*time*]
Selects files according to the dates on which they were last modified.

This qualifier is relevant only when used with the /BEFORE or the /SINCE qualifier. In addition, do not use /MODIFIED with /BACKUP or /EXPIRED.

If you omit a time value, TODAY is used. For more information on time specifications, see the *OpenVMS User's Manual*.

/MULTIPLE_FILES
Indicates that the *file-spec* parameter contains a list of file names to be checked. The *file-spec* file is opened and each record is read and treated as a *file-spec*.

/OUTPUT=*file-spec*
/NOOUTPUT
File name of the file in which to store readable MAC values. These MAC values represent both the file contents as well as the security settings. If you do not specify a file name, the default file name SYS$LOGIN:ENCRYPT$MAC.LIS is used.

/SECURITY=*file-spec*
/NOSECURITY
File name of the file in which to store binary MAC values. If you do not specify a file name, the default file name ENCRYPT$SEC.DAT is used.

Generates a MAC using the file's security settings: owner, protection settings, and optional ACL

/SINCE[=*time*]
Selects files that have a creation time before the time you specify.

If you omit *time*, TODAY is used. For more information on time specifications, see the *OpenVMS User's Manual*.

/UPDATE
/NOUPDATE
Associates new MAC values with one or more files.

ENCRYPT /CREATE_KEY

Creates a key definition to be used for encrypting files. The key is a string that represents the name under which the encryption key is stored in the key storage table.

Format

ENCRYPT /CREATE_KEY *key-name key-value* [*qualifiers*]

Parameters

key-name
Name under which the encryption key will be stored in the key storage table. Specify a character string, as follows:

- Valid length: 1 to 243 alphanumeric characters.
- Valid: dollar signs and underscores.
- Case sensitive: no.

Use a name that has meaning to you, to help you remember it.

Note

Key names beginning with ENCRYPT$ are reserved for Compaq.

key-value
String representing the value of the encryption key. Specify either ASCII text or a hexadecimal constant, as follows:

- ASCII text string (default)
  - Length: 8 to 240 characters.
  - The string is not case sensitive.
  - If you use characters other than alphanumeric characters, for example, blank spaces, enclose the string in quotation marks (" ").
- Hexadecimal constant
  - Use the /HEXADECIMAL qualifier.
  - Valid characters: 0 to 9, A to F.
  - Valid minimum length: 15 characters.
  - Do **not** enclose the value in quotation marks.

Qualifiers

/GROUP
Enters the key definition in the group key storage table.

/HEXADECIMAL
/NOHEXADECIMAL

Specifies that the value for the key is a hexadecimal number. Default: key values are interpreted as ASCII text characters (see the description of the *key-value* parameter).

/JOB

Enters the key definition in the job key storage table.

/LOG

Verifies successful creation of the key.

/PROCESS

Enters the key definition in the process key storage table.

/SYSTEM

Enters the key definition in the system key storage table.

Examples

1.

```
$ ENCRYPT /CREATE_KEY HAMLET
_ Key value: "And you yourself shall keep the key of it"
```

This command defines a key named HAMLET with character string value `And you yourself shall keep the key of it`.

2.

```
$ ENCRYPT /CREATE_KEY /HEXADECIMAL ARCANE 2F4A98F46BBC11D
```

This example defines a key named ARCANE with hexadecimal value 2F4A98F46BBC11D.

ENCRYPT /REMOVE_KEY

Deletes a key definition from a key storage table.

Format

ENCRYPT /REMOVE_KEY *key-name* [*qualifiers*]

Parameters

key-name
Key name previously stored in the key storage table with the ENCRYPT /CREATE_KEY command.

Qualifiers

/GROUP
Deletes the key definition from the group key storage table.

/JOB
Deletes the key definition from the job key storage table.

/PROCESS
Deletes the key definition from the process key storage table.

/SYSTEM
Deletes the key definition from the system key storage table.

B
Error Messages

The Encryption for OpenVMS commands can generate error and information messages. For descriptions of these messages and possible corrective actions, see the following sections:

- Messages produced by the ENCRYPT, ENCRYPT /CREATE_KEY, ENCRYPT /AUTHENTICATE, ENCRYPT /REMOVE_KEY, and DECRYPT commands (Section B.1)
- Messages produced by the Encryption option of the BACKUP utility (Section B.2)

B.1 ENCRYPT and DECRYPT Messages

The error messages documented in this section can be produced from the following commands:

- ENCRYPT /CREATE_KEY
- ENCRYPT /REMOVE_KEY
- ENCRYPT
- ENCRYPT /AUTHENTICATE
- DECRYPT

ALGONEWAY, algorithm is one-way and not suitable for file encryption,

Explanation: The specified algorithm and mode cannot be used to decrypt data. Encryption does not let you encrypt files without being able to decrypt them.

User Action: Choose a different algorithm or mode.

ALGNOTSPEC, algorithm name specification is a required parameter,

Explanation: You omitted the name of the algorithm in a call to an Encryption routine.

User Action: Specify a three-character long algorithm name.

ALGSUBNOT, algorithm submode option not supported,

Explanation: You supplied a submode parameter to an encryption primitive entry point that is not supported by the available implementation.

User Action: Verify that the contents of the algorithm control mask parameter are correct for the selected algorithm.

AUTHGEN, Authentication code generated for file: *file-spec*,

Explanation: A message authentication code (MAC) has been calculated for an existing file.

User Action: No action required.

AUTHMATCH, File *file-spec* successfully authenticated,

Explanation: The computed message authentication code (MAC) for the file matches the previously stored MAC.

User Action: No action required.

AUTHMISM, File authentication code mismatch for file: *file-spec*,

Explanation: The contents of the file have changed because the MAC does not match the one stored in the database. This may indicate file tampering.

User Action: First check the contents of the file, and then make sure that it belongs in your directory.

CONINIERR, unable to initialize the work area for the algorithm selected,

Explanation: The context and work area contain inconsistent check data and are assumed to be corrupted or incorrectly initialized.

User Action: Verify that the following took place:

- The proper sequence of initialize, encrypt or decrypt, and finalize calls was made when referencing the context area.
- The context area in allocated memory was not corrupted by other programming actions.

CONLENERR, context area length error,

Explanation: The context and work area supplied to the encryption primitive routine was not long enough for the encryption primitive routine to operate. This error is expected only if you make direct use of the encryption primitive entry points when the ENCRYPT$ library routines attempt to allocate the correct length of the context area from free memory.

User Action: Verify that the symbol used as the size of the context area is correct for the selected algorithm and that the resulting size reflects the minimum requirements for the encryption primitive routine.

CONNOTINI, context area not yet initialized,

Explanation: The Context Block has not been initialized by either ENCRYPT$INIT or ENCRYPT$K_FNC$INIT.

User Action: Check your program to be sure that the Context Block is initialized before an encryption or decryption operation.

CONPOIINI, context area pointer is already initialized or nonzero,

Explanation: An initialize call contained a nonzero context pointer.

User Action: Verify that the proper sequence of initialize, encrypt or decrypt, and finalize calls has referenced the pointer to the context area.

CRC FAIL, CRC comparison indicates tampered or corrupted file, *file-spec*,

Explanation: The cyclic redundancy check (CRC) value for the original plaintext file does not match the value of the decrypted file. Although the file has been saved, changes have occurred in the ciphertext file.

User Action: Do not rely on data in this file. Attempt to recover lost or corrupted information from backups or from the owner.

CRECONTIG, unable to create file as contiguous, *file-spec*,

Explanation: When a file is encrypted, file attributes are preserved in the encrypted file. One of the attributes specifies whether or not the original plaintext file was contiguous. When the file is decrypted, an attempt is made to create the output file with the same attribute. If the file cannot be created contiguously, this message results and a noncontiguous file is created.

User Action: If file contiguity is important, free up sufficient space on the output disk device so that it can contain the file in contiguous disk blocks. Purge files, delete unnecessary files, or select a different output device.

DBOPEN, Cannot open database file *file-spec*,

Explanation: Encryption could not access the database file you specified.

User Action: Check accompanying error messages for more information.

DBUNRDBL, Database is unreadable; check for correct key,

Explanation: You specified an incorrect encryption key. Only one key may be used for each database.

User Action: Use the correct key. Specify the same key you used when you originally updated the MAC.

DECRYPTED, file decrypted as specified,

Explanation: The file was decrypted as specified.

User Action: None.

ENCRYPTED, file encrypted as specified,

Explanation: The file was encrypted as specified.

User Action: None.

FILBADBLK, file contains bad blocks, processing not attempted, *file-spec*,

Explanation: You tried to encrypt a file containing bad blocks.

User Action: Ensure that any wildcard file specification does not include the name of the file that triggers this error. Do not delete the file before consulting with your system manager about how to recover the contents of the file or how to remove it from your directory.

FILDISKONLY, file encryption/decryption is supported for disk files only, *file-spec*,

Explanation: You attempted an encryption operation on either an input or output file that does not reside on a disk device, the only devices that are supported.

User Action: Copy the files to a disk device before attempting an encryption or decryption operation.

FILNODIR, file encryption of directories is not supported, *file-spec*,

Explanation: You attempted a file encryption operation on a directory file. The file encryption services are intended for user files only. The encryption and decryption of directories is not supported.

User Action: Ensure that any wildcard file specification does not include directory files. If you are at DCL level, use the /EXCLUDE=.DIR qualifier in the ENCRYPT or DECRYPT commands. If you are operating at the application program interface level, filter each file specification before calling ENCRYPT$ENCRYPT_FILE.

To encrypt whole directory structures, use the BACKUP utility with the /ENCRYPT qualifier.

FILNOPPF, file encryption of a process-permanent file is not supported, *file-spec*,

Explanation: You attempted a file encryption operation on a process-permanent file. Process-permanent files, even though they can reside on a disk as batch or log files, are presented to a process as though they were record devices and cannot be treated as disk files.

User Action: Reconstruct the batch or command file to copy input data to a temporary disk file before encrypting or decrypting the data to a disk file; then copy it to the output log as appropriate.

FILESTRUCT, input file structure error, *file-spec*,

Explanation: An internal logic error occurred when data from a compressed, encrypted file was decompressed.

User Action: Contact your Compaq support representative.

FILSTRUNS, structure of encrypted file is unsupported, *file-spec*,

Explanation: A file encrypted with the file encryption routine contains a routine version number to track any future enhancements of the file structure. If a file created by a later version of the software is presented to an earlier version for decryption, this message results.

Encryption is upward compatible — files encrypted with the current Encryption for OpenVMS version can be decrypted using a later version. But, the reverse is not necessarily always possible.

This error can also indicate an attempt to decrypt a file using the incorrect key.

User Action: None. The file cannot be decrypted with the current software version.

HIGHVER, creating output file for which higher versions exist, *file-spec*,

Explanation: When creating an output file during file encryption or decryption, you supplied an output file specification that forced the creation of a file with a version number lower than another file in the directory.

User Action: If this is not the intended action, provide a file specification that does not force a version number value with the /OUTPUT=filename qualifier.

ILLALGMOD, algorithm submode selection unknown or unsupported,

Explanation: This error indicates that you specified a submode code that the indicated algorithm does not support. This error results only if there is a parameter error in a direct call to an encryption primitive routine.

User Action: Verify that the algorithm selected makes use of the submode code supplied.

ILLALGSEL, algorithm selection unknown or unsupported,

Explanation: You supplied an algorithm code to a function that is not supported or installed on this system.

User Action: None. That algorithm may not be used in this installation.

ILLDESTYP, illegal descriptor type for specifying parameter,

Explanation: You passed a descriptor address as a parameter to the routine returning this error. The descriptor type field indicates that this descriptor type is not supported for passing of parameters.

User Action: Verify that the descriptors passed conform to the specified type requirements. It may be necessary to explicitly initialize the descriptor type field to avoid default or uninitialized values.

IMGVERNEQ, algorithm image version is no longer supported,

Explanation: An upgrade to a newer version of the software is incomplete. Former Encryption for OpenVMS images remain on your system.

User Action: Re-install the Encryption for OpenVMS software.

INCKEYDEF, incompatible key definition,

Explanation: The specified key does not meet the requirements of the specified algorithm.

User Action: Select a different key value or, if the key value has been randomly generated as part of the user application, generate another value.

INPLENERR, input length does not meet algorithm requirements,

Explanation: The input data length is not valid. The DESECB and DESCBC modes require input data that is a multiple of 8 bytes in length. The basic algorithm operates only on 64 bits of data in each pass. Ensure that the input data length is a multiple of 8 bytes.

User Action: Revise the input data length to be a multiple of 8 bytes.

INSCONSPA, insufficient context space to support this algorithm's needs,

Explanation: The ENCRYPT$INIT function attempted to allocate space from dynamic memory for a buffer to contain the encryption stream context and work area. This memory allocation failed.

User Action: Increase the process parameters to permit more virtual memory, or reconstruct your application to leave more virtual memory available for system library and encryption functions.

INSTALLIT, key translation error in Encryption for OpenVMS indicates that product installation may not be complete,

Explanation: The Encryption startup procedure SYS$STARTUP:ENCRYPT$START.COM has not executed.

User Action: Execute the startup procedure from the system manager's account.

INVARGCOU, invalid argument count for ENCRYPT$ routine,

Explanation: You did not supply enough arguments to one of the Encryption for OpenVMS library routines to initiate the indicated function.

User Action: Verify the call format and specify the correct number of arguments.

INVARGVAL, invalid argument value and/or count,

Explanation: The routine issuing this message was called with an invalid argument count or value.

User Action: Verify the callable routine sequences.

INVFLAGS, invalid options flags specified,

Explanation: Invalid option flag bits were set in the flags argument to an Encryption routine.

User Action: Correct the program to properly initialize unused bits in the flags argument longword to zero.

INVROUNDUP, invalid algorithm block buffer roundup specification,

Explanation: An internal error has occurred.

User Action: Contact your Compaq support representative.

INVWEAK_KEY, key rejected; use of weak key for file encryption is invalid,

Explanation: A weak key was specified for a file encryption operation.

User Action: Specify a different key value.

KEYBUFCKS, checksum of encrypted file key is invalid,

Explanation: The checksum of the stored random key (under which the file is actually encrypted) is incorrect following decryption using the specified decryption key. This usually means that the incorrect key has been specified.

User Action: Determine the correct key with which to decrypt the file.

KEYLENERR, key length does not meet algorithm requirements,

Explanation: The key length does not contain enough characters. The DES algorithm requires a minimum of 8 bytes for its key string. Other algorithms may have other requirements.

User Action: Redefine a key containing more characters.

KEYPARERR, key parity error,

Explanation: The DES algorithm requires that the key string extracted from key storage have odd-bit parity in each byte. In normal operating mode, the algorithm forces odd parity before using the key. Under certain conditions, the encryption primitive routine can be called directly with parameters that suppress the forcing of odd parity. In that case, if the key has incorrect parity, this error will be returned.

User Action: Revise the DES call parameter to force generation of odd parity, or reinsert the key string into key storage and indicate that odd parity is to be set.

KEYTRNERR, unable to obtain key value from key storage,

Explanation: You supplied a key name that is not found in the key storage table.

User Action: Verify that the key is defined as intended and that the name is supplied to the initialize function correctly.

KEYUNKNOW, key name unknown,

Explanation: You specified a key incorrectly.

User Action: Verify the key name (for example, check the spelling) and specify it correctly.

NEWDB, new authentication code database has been created,

Explanation: A new database is created to store binary message authentication code (MAC) values.

User Action: No action required.

NEWSECDB, New authentication security settings database has been created,

Explanation: A new security database is created to store binary message authentication code (MAC) values.

User Action: None.

NODECRYPT, decrypt operations are not permitted on this context/stream,

Explanation: Decryption is not permitted when the context has been initialized for encryption.

User Action: Reinitialize the context to permit decryption.

NOENCRYPT, encrypt operations not permitted on this context/stream,

Explanation: Encryption is not permitted when the context has been initialized for decryption.

User Action: Reinitialize the context to permit encryption.

NOENTRY, file *file-spec* does not appear in the authentication database,

Explanation: The file does not have an associated message authentication code (MAC) stored in the MAC database. The file is either new, renamed, or has not been associated with a MAC.

Sometimes this message is an indication of file tampering.

User Action: Determine whether the file belongs in this database.

NOKGENROU, no key generation routine is provided for this algorithm,

Explanation: The specified algorithm did not contain a random key generation routine.

User Action: Contact your Compaq support representative.

NOKTSTROU, no key filter routine is provided for this algorithm,

Explanation: The specified algorithm did not contain a key filter routine.

User Action: Contact your Compaq support representative.

NOTAUTHDB, file *file-spec* is not an authentication database,

Explanation: The file you specified is not a database created by the Encryption for OpenVMS software. It is not usable as an authentication database.

User Action: Use a different file specification.

NOTDEL, error prevents deletion of file *file-spec*,

Explanation: You specified the /DELETE qualifier when encrypting or decrypting a file, but you lack delete access to this file.

User Action: Change the file protection and delete the file using the DCL DELETE command.

NOTESTROU, no test routine is available for this algorithm,

Explanation: The specified algorithm did not contain a test routine.

User Action: Contact your Compaq support representative.

NOTHEXVAL, key value not hexadecimal constant,

Explanation: You specified a key value that is not a hexadecimal constant with ENCRYPT /CREATE_KEY /HEXADECIMAL.

User Action: Either remove the /HEXADECIMAL qualifier or ensure that the key value string is composed of digits in the range 0 to 9 and A to F.

NOTSECDB, Setting is not in security database,

Explanation: The file you specified is not a security database created by the Encryption for OpenVMS software. It is not usable as a security authentication database.

User Action: Validate that the file specification is correct. The Encryption for OpenVMS software creates the file ENCRYPT$MAC_SEC.DAT in the SYS$LOGIN directory by default.

NOTYETIMP, this function is not yet implemented,

Explanation: The call requested a function that has not been implemented.

User Action: Contact your Compaq support representative.

OUTLENERR, output length does not meet algorithm requirements,

Explanation: You did not supply an output buffer long enough to hold the output from the encryption or decryption operation. Because some algorithms increase or decrease data-byte count, check the requirements of the different algorithms.

User Action: Supply a larger output buffer.

PARSEFAIL, error parsing *file-spec*,

Explanation: Encryption could not locate the file you specified in an ENCRYPT or a DECRYPT command.

User Action: Check the file name that you specified. An accompanying RMS message gives additional information about the error. Re-enter the command using the correct file name.

SECAUTHGEN, Security authentication code generated for *filename*,

Explanation: A message authentication code (MAC) has been calculated for the specified file based on the file's security settings.

User Action: None.

SECAUTHMATCH, Security settings for *filename* successfully authenticated,

Explanation: The computed message authentication code (MAC) for the file matches the previous stored MAC in the security database.

User Action: None.

SECAUTHMISM, Security authentication code mismatch for file *filename*,

Explanation: The security settings of the file have changed because the message authentication code (MAC) does not match the MAC stored in the security database. This may indicate security settings tampering.

User Action: Perform a $ DIRECTORY/SECURITY on the file to validate the file has the proper security settings.

SECNOENTRY, Security entry for *filename* does not appear in security database,

Explanation: The file does not have an assoicated message authentication code (MAC) stored in the security database. The file is either new, renamed, or has not been associated with a MAC. Sometimes this message is an indication of file tampering.

User Action: Determine whether the file belongs in this database.

SECSUMM1, Summary: Security settings authenticated: *n*,

Explanation: Lists the number of files whose message authentication codes (MACs) match previously stored MACs.

User Action: None.

SECSUMM2, Security settings failing authentication: *n*,

Explanation: Lists the number of files whose message authentication codes (MACs) do not match previously stored MACs.

User Action: None.

SECSUMM3, Security settings not in database: *n*,

Explanation: Lists the number of files with no associated message authentication codes (MACs).

User Action: None.

SECUPDENT, Authentication code for security settings of file *filename* has been updated,

Explanation: The message authentication code (MAC) based upon the security settings of the file have been updated with a new MAC in the security database.

User Action: None.

STATISTICS, encryption stream statistics,

Explanation: This message precedes the display of encryption statistics when the /SHOW qualifier has been specified with either the STATISTICS or the ALL keyword.

User Action: None. This is a success message.

SUMMARY1, Files successfully authenticated: *n*,

Explanation: Lists the number of files whose message authentication codes (MACs) match previously stored MACs.

User Action: No action required.

SUMMARY2, Files failing authentication: *n*,

Explanation: Lists the number of files whose MACs do not match previously stored MACs.

User Action: No action required.

SUMMARY3, Files not in database: *n*,

Explanation: Lists the number of files with no associated MACs.

User Action: No action required.

TESTFAIL, test failed. Test Number: *n*,

Explanation: One of the tests for the encryption primitive routine failed, indicating that the algorithm is not operating correctly.

User Action: Contact the supplier of the algorithm.

UNSFTR, Feature *feature-name*, written by product version *version-number* is not supported,

Explanation: Encryption is unable to decrypt the specified file correctly. When the severity is W, the file is decrypted, but a processing feature is omitted. When the severity is E, the file is not decrypted.

User Action: Decrypt the file on a system running the Encryption for OpenVMS version displayed in the error message. Or, upgrade to the current version of Encryption for OpenVMS.

UNSAGTFMT, algorithm dispatch table format is not supported,

Explanation: An upgrade to a newer version of Encryption for OpenVMS is incomplete. Former Encryption images remain on your system.

User Action: Re-install the Encryption fpr OpenVMS software.

UPDENTRY, authentication code for file *file-spec* has been updated,

Explanation: The message authentication code (MAC) in the database file is updated to a new MAC.

User Action: No action required.

UPDSECENT, Security authentication code for file *filename* has been updated,

Explanation: A new MAC, based upon the security settings of the file that was specified, has been created and stored in the security settings database.

User Action: None.

WEAK_KEY, key value is rejected by key filter as weak or incompatible,

Explanation: The specified key value is rejected as a weak key by the encryption primitive routine.

User Action: You can choose to use the weak key for encryption or you can specify a different key value.

B.2 BACKUP Utility Messages

The BACKUP /ENCRYPT command can produce the following messages.

BACNOTENC, backup set is not encrypted. Ignoring /ENCRYPT qualifier,

Explanation: The save set is not encrypted.

User Action: None. The save set is restored without decryption.

DECRYPERR, error encountered in encryption facility while decrypting a backup record,

Explanation: An error has occurred decrypting save-set data. The accompanying message gives additional information about the error.

User Action: Depends on the specific error.

ENCALGNOT, specified encryption algorithm is not supported in this copy of BACKUP,

Explanation: You specified an unsupported encryption algorithm to the algorithm keyword.

User Action: Re-enter the command, using a supported encryption algorithm.

ENCBSRNOT, internal error. BACKUP Summary, Record required for decryption not found,

Explanation: The BACKUP Summary Record that contains the data key under which the save set is encrypted was not found at the start of the save set. It is possible that the save set was not encrypted.

User Action: None. The save set is probably corrupted and cannot be restored. If the save set was in fact not encrypted, it can be recovered by omitting the /ENCRYPT qualifier.

ENCFINERR, error encountered closing an encryption stream,

Explanation: An error has occurred cleaning up the encryption context. The accompanying message gives additional information about the error.

User Action: Depends on the specific error.

ENCINIERR, an error was encountered initializing an encryption stream,

Explanation: An error has occurred initializing the encryption operation. The accompanying message gives additional information about the error.

User Action: Depends on the specific error.

ENCKEYMAT, the supplied decryption key does not yield a readable save set,

Explanation: The checksum on the decrypted data key and initialization vector does not match the checksum that was stored previously. This usually means that an incorrect key name or value has been specified.

User Action: Determine the correct key to decrypt the save set.

ENCNOTSUP, encryption of save sets is not supported in this copy of BACKUP,

Explanation: This version of BACKUP does not support encryption.

User Action: Use another version of BACKUP that supports encryption.

ENCQUAIGN, encryption not supported for this copy operation. Ignoring /ENCRYPT qualifier,

Explanation: BACKUP supports encryption only in save sets, not in file-to-file or volume-to-volume copy operations.

User Action: Re-enter the command, either omitting the /ENCRYPT qualifier or using a save set, as your needs determine.

ENCRYPERR, error encountered in encryption facility while encrypting a backup record,

Explanation: An error has occurred encrypting save-set data. The accompanying message gives additional information about the error.

User Action: Depends on the specific error.

ENCSAVSET, save set is encrypted. /ENCRYPT must be specified,

Explanation: The save set is encrypted and, therefore, cannot be read.

User Action: Decrypt the save set. Specify the correct key with the /ENCRYPT qualifier.

KEYLENERR, entered key value is too long,

Explanation: The key specified is too long for the encryption algorithm.

User Action: Re-enter the command using a shorter key.

KEYNOTVER, key value not entered identically as verification,

Explanation: The key values entered to the key and verification prompts are different.

User Action: Re-enter the keys, taking care to type the same value both times.

Index

A

Algorithm
  used by Encryption, 1–1
Application programming
  See Routines
Authentication
  definition of, 1–3
  ENCRYPT/AUTHENTICATE command
    syntax, A–9
  overview, 1–3

B

BACKUP utility, 3–24 to 3–26
  BACKUP/ENCRYPT command, 3–24
    DESCBC algorithm, 3–24
    syntax, A–2
  /SAVE_SET qualifier, 3–24
  standalone, 3–24

C

Callable routines
  See Routines
Calling Standard
  requirements, 4–2
Ciphertext
  definition of, 1–1
Commands
  See also individual commands
  syntax descriptions, A–1 to B–1
  tutorial, 3–1 to 3–26

D

Data Encryption Standard
  See DES
DCL commands
  See Commands
  See individual commands
DECRYPT command
  format, 3–17
  purpose, 3–17
  syntax, A–3
Decryption
  definition of, 1–2
  prerequisite, 1–2, A–14
  requirements, 3–4, 3–17
DES
  documented in FIP, 1–1
  implemented in Encryption for OpenVMS, 1–1
  modes, 3–10

E

ENCRYPT$MAC.LIS
  for storing MAC values, 3–15
ENCRYPT/AUTHENTICATE command
  syntax, A–9
ENCRYPT/CREATE_KEY command
  syntax, A–12
  verifying, 3–2
ENCRYPT/REMOVE_KEY command
  syntax, A–14
ENCRYPT command
  syntax, A–6
Encryption
  definition of, 1–1
  ENCRYPT command, 3–4
  of save sets, 3–24, A–2
  prerequisite, 1–1, 3–4, A–6, A–12
Encryption process
  overview, 1–1 to 1–4
  steps in, 1–1
  tutorial, 3–1 to 3–26

F

Federal Information Processing Standard
  See FIP
FIP
  encryption standard, 1–1

I

Installation, 2–1 to 2–6
  from CD, 2–3
  post-installation tasks, 2–5
  pre-installation tasks, 2–1
  requirements, 2–1
  running, 2–3

Interfaces
  callable routines, 1–3, 4–1 to A–1
  DCL commands, 1–3, 3–1 to 3–26

K

Key
  creating
    ENCRYPT/CREATE_KEY command
      syntax, A–12
  data key
    definition of, 1–1
  definition of, 1–1
  deleting
    ENCRYPT/REMOVE_KEY command
      syntax, A–14
  functions of, 1–1
  used with decryption, 1–2
  used with encryption, 1–1
  weak
    avoiding, 3–2
    definition of, 3–2

L

License registration, 2–2
LMF
  registering your license, 2–2

M

MAC
  calculated, 1–3, 3–13
  file contents, 3–13, 3–14
  security, 3–13, 3–14
  specifying database, 3–15
  used by ENCRYPT/AUTHENTICATE command, A–9
  values and status stored, 3–15
Message Authentication Code
  See MAC

N

NBS
  DES documentation, 1–1

O

OpenVMS Calling Standard
  requirements, 4–2

P

PAK
  registering, 2–2
Plaintext
  definition of, 1–1
POLYCENTER Software Installation procedure, 2–1
Product overview, 1–1 to 1–4
Programming
  See Routines

R

Routines
  See also individual routines
  ENCRYPT$DECRYPT, 4–6
  ENCRYPT$DECRYPT_ONE_RECORD, 4–8
  ENCRYPT$DEFINE_KEY, 4–10
  ENCRYPT$DELETE_KEY, 4–12
  ENCRYPT$ENCRYPT, 4–13
  ENCRYPT$ENCRYPT_FILE, 4–15
  ENCRYPT$ENCRYPT_ONE_RECORD, 4–18
  ENCRYPT$FINI, 4–20
  ENCRYPT$GENERATE_KEY, 4–21
  ENCRYPT$INIT, 4–23
  ENCRYPT$STATISTICS, 4–26
  overview, 4–1 to 4–5
  programming interface, 1–3
  syntax descriptions, 4–5 to A–1

S

Save sets
  encrypting, 3–24, A–2
/SAVE_SET qualifier
  of BACKUP command, 3–24
Security
  ACL-based, 1–1
  encryption-based, 1–1
  UIC-based, 1–1
Syntax
  See Commands
  See Routines
SYS$LOGIN:ENCRYPT$MAC.LIS
  for storing MAC values, 3–15

W

Weak keys
  See Key