Encryption Facility for z/OS:

User's Guide

Version 1 Release 10

SA23-1349-03

Encryption Facility for z/OS:

User's Guide

Version 1 Release 10

SA23-1349-03

Note
Before using this information and the product it supports, read the information under “Notices,” on page 69.

Fourth Edition (June 2006)

This edition applies to version 1, release 1, modification 0 of IBM Encryption Facility for z/OS (product number 5655-P97) and to all subsequent releases and modifications until otherwise indicated in new editions. This is a major revision of SA23-1349-02.

IBM welcomes your comments. A form for readers' comments may be provided at the back of this document, or you may address your comments to the following address:

International Business Machines Corporation
Department 55JA, Mail Station P384
2455 South Road
Poughkeepsie, NY 12601-5400
United States of America

FAX (United States & Canada): 1+845+432-9405
FAX (Other Countries):
Your International Access Code +1+845+432-9405

IBMLink™ (United States customers only): IBMUSM10(MHVRCFS)
Internet e-mail: mhvrcfs@us.ibm.com
World Wide Web: www.ibm.com/servers/eserver/zseries/zos/webqs.html

If you would like a reply, be sure to include your name, address, telephone number, or FAX number.

Make sure to include the following in your comment or note:
- Title and order number of this document
- Page number or topic related to your comment

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Contents

Figures

About This Book

This information supports z/OS® (5694–A01). The document contains information about using IBM Encryption Facility for z/OS (Encryption Facility).

Encryption Facility provides encryption and decryption processing of data for exchange between different systems and platforms and for archiving purposes. It makes use of hardware compression and encryption and relies on a centralized key management based on the z/OS Integrated Cryptographic Service Facility (ICSF). Encryption Facility consists of the following optional features:

- Encryption Facility Encryption Services for z/OS
- Encryption Facility DFSMSdss Encryption

A licensed Java™ reference program called Encryption Facility for z/OS Client is also downloadable from the Worldwide Web.

This document provides you with the information to set up the encryption and decryption through job control entry (JCL) statements on z/OS for the Encryption Services and how to use the Security Server RACF® commands with the product. It also provides an overview of Encryption Facility for z/OS Client and DFSMSdss Encryption.

Who should read this book

Anyone who plans, installs, customizes, administers, and uses Encryption Facility should use this document. It should also be used by those who install, configure, or provide support in the following areas:

- z/OS
- Integrated Cryptographic Service Facility (ICSF)
- Security Server Resource Access Control Facility (RACF)
- DFSMSdss

This product assumes that you have experience installing, configuring, and using z/OS, ICSF, RACF, and DFSMSdss. If you plan to use the Java reference code, the product assumes that you understand Java–related concepts and tasks. You should be knowledgeable about the following areas:

- Public key infrastructure (PKI) technology
- Use of Job Entry Control (JCL) language
- Use of RACF commands and interfaces
- Use of DFSMSdss commands and interfaces

How to use this book

This document contains the following chapters:

- Chapter 1, “Overview of IBM Encryption Facility for z/OS,” on page 1 presents an overview of Encryption Facility, the functions of the product, and hardware and software requirements.
- Chapter 2, “Getting started,” on page 11 presents information about installation and getting started with Encryption Facility.
- Chapter 3, “Encrypting files through CSDFILEN of Encryption Services,” on page 15 presents information about using the CSDFILEN batch program for the Encryption Services.

- Chapter 4, “Decrypting files through CSDFILDE of the Encryption Services,” on page 31 presents information about using the CSDFILDE batch program for the Encryption Services.
- Chapter 5, “Using Encryption Facility for z/OS Client,” on page 39 presents overview information about using Encryption Facility for z/OS Client and how to obtain more information.
- Chapter 6, “Using DFSMSdss Encryption,” on page 43 presents overview information about using DFSMSdss Encryption and how to obtain more information.
- Chapter 7, “Using RACF with Encryption Facility,” on page 47 presents information about using RACF with Encryption Facility.
- Chapter 8, “User scenarios,” on page 57 presents user scenarios for Encryption Facility.

Where to find more information

Where necessary this document references information in other documents. For complete titles and order numbers for all elements of z/OS, see *z/OS Information Roadmap*.

Related information:

- Integrated Cryptographic Service Facility (ICSF) publications
- Security Server RACF publications

Related publications

The Encryption Facility library contains the following books:

- *IBM Encryption Facility for z/OS: Licensed Program Specifications*
- *IBM Encryption Facility for z/OS: Program Directory*
- *IBM Encryption Facility for z/OS: User's Guide*

The following ICSF books are referenced in this book:

- *z/OS Cryptographic Services ICSF Administrator's Guide*
- *z/OS Cryptographic Services ICSF Application Programmer's Guide*
- *z/OS Cryptographic Services ICSF Overview*
- *z/OS Cryptographic Services ICSF System Programmer's Guide*

The following RACF book is referenced in this book:

- *z/OS Security Server RACF Command Language Reference*

The following DFSMS books are referenced in this book:

- *z/OS DFSMSdfp Advanced Services*
- *z/OS DFSMS Storage Administration Reference*.
- *z/OS DFSMShsm Storage Administration Guide*
- *z/OS DFSMShsm Implementation and Customization Guide*

Documentation for the PCI Cryptographic Coprocessor is found on the web at http://www.ibm.com/security/cryptocards/library.shtml.

- *IBM® 4758 PCI Cryptographic Coprocessor CCA Support Program Installation Manual for IBM 4758 Models 002 and 023*

- *IBM 4758 PCI Cryptographic Coprocessor CCA Basic Services Reference and Guide for the IBM 4758 Models 002 and 023*
- *IBM 4758 PCI Cryptographic Coprocessor General Information*
- *IBM 4758 PCI Cryptographic Coprocessor Installation*

Using LookAt to look up message explanations

LookAt is an online facility that lets you look up explanations for most of the IBM messages you encounter, as well as for some system abends and codes. Using LookAt to find information is faster than a conventional search because in most cases LookAt goes directly to the message explanation.

You can use LookAt from these locations to find IBM message explanations for z/OS elements and features, z/VM®, VSE/ESA™, and Clusters for AIX® and Linux™:

- The Internet. You can access IBM message explanations directly from the LookAt Web site at www.ibm.com/servers/eserver/zseries/zos/bkserv/lookat/.
- Your z/OS TSO/E host system. You can install code on your z/OS or z/OS.e systems to access IBM message explanations using LookAt from a TSO/E command line (for example: TSO/E prompt, ISPF, or z/OS UNIX® System Services).
- Your Microsoft® Windows® workstation. You can install LookAt directly from the *z/OS Collection* (SK3T-4269) or the *z/OS and Software Products DVD Collection* (SK3T-4271) and use it from the resulting Windows graphical user interface (GUI). The command prompt (also known as the DOS > command line) version can still be used from the directory in which you install the Windows version of LookAt.
- Your wireless handheld device. You can use the LookAt Mobile Edition from www.ibm.com/servers/eserver/zseries/zos/bkserv/lookat/lookatm.html with a handheld device that has wireless access and an Internet browser (for example: Internet Explorer for Pocket PCs, Blazer or Eudora for Palm OS, or Opera for Linux handheld devices).

You can obtain code to install LookAt on your host system or Microsoft Windows workstation from:

- A CD-ROM in the *z/OS Collection* (SK3T-4269).
- The *z/OS and Software Products DVD Collection* (SK3T-4271).
- The LookAt Web site (click **Download** and then select the platform, release, collection, and location that suit your needs). More information is available in the LOOKAT.ME files available during the download process.

Using IBM Health Checker for z/OS

IBM Health Checker for z/OS is a z/OS component that installations can use to gather information about their system environment and system parameters to help identify potential configuration problems before they impact availability or cause outages. Individual products, z/OS components, or ISV software can provide checks that take advantage of the IBM Health Checker for z/OS framework. This book refers to checks or messages associated with this component.

For additional information about checks and about IBM Health Checker for z/OS, see *IBM Health Checker for z/OS: User's Guide*. Starting with z/OS V1R4, z/OS users can obtain the IBM Health Checker for z/OS from the z/OS Downloads page at www.ibm.com/servers/eserver/zseries/zos/downloads/.

SDSF also provides functions to simplify the management of checks. See *z/OS SDSF Operation and Customization* for additional information.

Other sources of information

IBM provides customer-accessible discussion areas where PKI Services and RACF may be discussed by customer and IBM participants. Other information is also available through the Internet.

IBM discussion area

IBM provides the *ibm.servers.mvs.racf* newsgroup for discussion of PKI Services and RACF-related topics. You can find this newsgroup on news (NNTP) server *news.software.ibm.com* using your favorite news reader client.

Internet sources

The following resources are available through the Internet to provide additional information about PKI Services, RACF, and many other security-related topics:

- **Online library**

  To view and print online versions of the z/OS publications, use this address:www.ibm.com/servers/eserver/zseries/zos/bkserv/
- **Redbooks™**

  The Redbooks™ that are produced by the International Technical Support Organization (ITSO) are available at the following address:www.ibm.com/redbooks
- **Enterprise systems security**

  For more information about security on the zSeries® platform and z/OS, use this address: www.ibm.com/servers/eserver/zseries/zos/security/
- **PKI Services home page**

  You can visit the PKI Services home page on the World Wide Web using the following address. Check this site for updates regarding PKI Services. http://www.ibm.com/servers/eserver/zseries/zos/pki/.
- **Techdocs**

  You can visit the Techdocs - Technical Sales Library home page on the World Wide Web using the following address. Use the search keyword “crypto” to help narrow your search: www.ibm.com/support/techdocs/.
- **RACF home page**

  You can visit the RACF home page on the World Wide Web using the following address. www.ibm.com/servers/eserver/zseries/zos/racf/goodies.html .
- **RACF-L discussion list**

  Customers and IBM participants may also discuss RACF on the RACF-L discussion list. RACF-L is not operated or sponsored by IBM; it is run by the University of Georgia.

  To subscribe to the RACF-L discussion and receive postings, send a note to:

  ```
  listserv@listserv.uga.edu
  ```

  Include the following line in the body of the note, substituting your first name and last name as indicated:

  ```
  subscribe racf-l first_name last_name
  ```

  To post a question or response to RACF-L, send a note, including an appropriate Subject: line, to:

racf-l@listserv.uga.edu

- **RACF sample code**

You can get sample code, internally-developed tools, and exits to help you use RACF. This code works in our environment, at the time we make it available, but is not officially supported. Each tool or sample has a README file that describes the tool or sample and any restrictions on its use.

To access this code from a Web browser, go to the RACF home page and select the "Downloads" topic from the navigation bar, or go to ftp://ftp.software.ibm.com/eserver/zseries/zos/racf/.

The code is also available from `ftp.software.ibm.com` through anonymous FTP. To get access:

1. Log in as user anonymous.
2. Change the directory, as follows, to find the subdirectories that contain the sample code or tool you want to download:

   ```
   cd eserver/zseries/zos/racf/
   ```

An announcement will be posted on RACF-L discussion list and on newsgroup *ibm.servers.mvs.racf* whenever something is added.

Note: Some Web browsers and some FTP clients (especially those using a graphical interface) might have problems using `ftp.software.ibm.com` because of inconsistencies in the way they implement the FTP protocols. If you have problems, you can try the following:

- Try to get access by using a Web browser and the links from the RACF home page.
- Use a different FTP client. If necessary, use a client that is based on command line interfaces instead of graphical interfaces.
- If your FTP client has configuration parameters for the type of remote system, configure it as UNIX instead of MVS™.

Restrictions

Because the sample code and tools are not officially supported,

– There are no guaranteed enhancements.

– No APARs can be accepted.

Do you have problems, comments, or suggestions?

Your suggestions and ideas can contribute to the quality and the usability of this book. If you have problems while using this book, or if you have suggestions for improving it, complete and mail the Reader's Comment Form found at the back of the book.

Summary of changes

Summary of changes
for SA23-1349-03
Encryption Facility for z/OS Version 1 Release 1

New Information: This release of *IBM Encryption Facility for z/OS: User's Guide* contains a reference to the Web site with information about downloading and using IBM Decryption Client for z/OS (Decryption Client).

This document contains terminology, maintenance, and editorial changes. Technical changes or additions to the text and illustrations are indicated by a vertical line to the left of the change.

Summary of changes
for SA23-1349-02
Encryption Facility for z/OS Version 1 Release 1

New Information: This release of *IBM Encryption Facility for z/OS: User's Guide* contains information about specifying multiple RSA control statements for Encryption Facility Services and multiple RSA certificate aliases for Encryption Facility for z/OS Client. Encryption Facility for z/OS Client also supports fixed records for encryption. In addition this document provides overview information about using ICSF utility panels to manage RSA public/private key pairs in the ICSF public key data set (PKDS).

This document contains terminology, maintenance, and editorial changes. Technical changes or additions to the text and illustrations are indicated by a vertical line to the left of the change.

Summary of changes
for SA23-1349-01
Encryption Facility for z/OS Version 1 Release 1

New Information: This release of Encryption Facility contains information for DFSMShsm™ and the DFSMSdss Encryption feature. For complete information about using DFSMShsm with the DFSMSdss Encryption feature, see the following publications:

- *z/OS DFSMShsm Storage Administration Guide*
- *z/OS DFSMShsm Implementation and Customization Guide*

Chapter 1. Overview of IBM Encryption Facility for z/OS

This chapter presents an overview of IBM Encryption Facility for z/OS (Encryption Facility), the functions of the product, and hardware and software requirements.

What is Encryption Facility?

The need for creating secure archived copies of business data is a critical security concern. Encrypting data that can be recovered at any time offers a high degree of privacy protection from unwanted access. Encryption Facility provides this protection by offering encryption of data for exchange between different systems and platforms and for archiving purposes. It makes use of hardware compression and encryption and relies on a centralized key management based on the z/OS Integrated Cryptographic Service Facility (ICSF) that is highly secure and easy to use.

Encryption Facility makes use of ICSF to perform encryption and decryption and to manage cryptographic keys. To encrypt data files Encryption Facility uses the following kinds of cryptographic keys:

- TDES triple-length keys
- 128–bit AES keys

For information about cryptographic keys, see the following publications:

- *z/OS Cryptographic Services ICSF Administrator's Guide*
- *z/OS Cryptographic Services ICSF Application Programmer's Guide*

For information about hardware requirements for Encryption Facility, see "Hardware and software requirements" on page 7.

Features available with Encryption Facility

Version 1 Release 1.0 of IBM Encryption Facility for z/OS provides the following optional features:

Table 1. Features for Encryption Facility

Feature	Description
IBM Encryption Facility for z/OS Encryption Services, called in this document **Encryption Services**	Complementary encryption and decryption batch programs that run on z/OS and allow you to encrypt and decrypt data
IBM Encryption Facility for z/OS DFSMSdss Encryption, called in this document **DFSMSdss Encryption**	Services that run on z/OS DFSMSdss™ and allow you to use DFSMSdss commands to encrypt and decrypt data. With this feature, you can also use DFSMShsm commands to encrypt and decrypt data.

IBM Encryption Facility for z/OS Encryption Facility for z/OS Client, called in this document **Encryption Facility for z/OS Client**, is a Java reference program that allows you to encrypt and decrypt z/OS format data on non-z/OS platforms. Encryption Facility for z/OS Client is available as licensed downloadable code from the Web. See "Software requirements" on page 9.

Encryption Services

Through Encryption Services, you can encrypt data on DASD or tape into a file that contains a header with enough information to recover or decrypt the data.

Moreover, the encrypted file and its associated encryption key can withstand multiple changes to the master key in the ICSF hardware on the original or target systems. As a result, you can recover archived files that you create through Encryption Facility at any time, even years after the files are created.

You code job control statements (JCL) to control the following Encryption Services functions:

- CSDFILEN to encrypt the data
- CSDFILDE to decrypt the data

You can optionally specify that the data is to be compressed before encryption and that the encrypted data, which is an output sequential file, is to be sent to tape for archiving or transfer.

Encryption Facility for z/OS Client

Encryption Facility for z/OS Client is Java reference code that allows you to control encryption and decryption of data on platforms other than z/OS or on z/OS systems that do not use Encryption Services. In that way, you can exchange encrypted data between z/OS and non-z/OS platforms as long as the encrypted data is created through Encryption Services or Encryption Facility for z/OS Client.

DFSMSdss Encryption

DFSMSdss Encryption is an optional feature that allows you to use z/OS DFSMSdss to encrypt data from the DFSMSdss DUMP command for tape or DASD. You can decrypt the data through the RESTORE command. With this feature you can also use the DFSMShsm dump class settings to encrypt data dumped through the BACKVOL DUMP command and automatic dump processing and decrypt the data through the RECOVER command.

Comparison of Encryption Facility features and functions

Table 2 on page 3 summarizes the functions for Encryption Services, DFSMSdss Encryption, and Encryption Facility for z/OS Client:

Table 2.

Encryption Services	DFSMSdss Encryption	Encryption Facility for z/OS Client
• Allows encryption and compression of data files • Supports decryption and decompression of data files • Makes use of z/OS centralized key management and access authentication • Uses IBM mainframe server cryptographic and compression capabilities • Can use either public/private key pairs or passwords to create secure exchange between partners.	• Allows encryption and compression of dump data sets created by DFSMSdss or DFSMShsm • Supports decryption and decompression during RESTORE process for DFSMSdss and RECOVER for DFSMShsm • Makes use of z/OS centralized key management and IBM mainframe cryptographic functions and compression • Can use either public/private key pairs or passwords to create secure exchange between partners.	• Optional feature that you can order from the World Wide Web • Java-based code that allows client systems to decrypt and encrypt tapes for exchange with z/OS systems • Requires a license from the Web site to be used in conjunction with the Encryption Services feature on z/OS systems • Can be used on any Java-enabled system • Can use either public/private key pairs or passwords to create secure exchange between partners. **Decryption Client:** You can also download free decryption services for Encryption Facility for z/OS Client called Decryption Client. For information see the following Web site: http://www.ibm.com/servers/eserver/zseries/zos/downloads/#asis

Security Server RACF enhancements

Enhancements to Security Server RACF (RACF), an element of z/OS, allow you to store internal RSA public and private keys for encrypted data in the ICSF public key data set (PKDS) and specify the PKDS labels for security certificates associated with encrypted data. Although you can use your own programs to load the PKDS, RACF provides a command (RACDCERT) to load the keys.

Summary of Encryption Facility functions

Table 3 summarizes the functions and support that Encryption Facility for z/OS provides:

Table 3.

Support or function	Available with IBM Encryption Facility
z/OS and z/OS.e	V1.4 and later releases (see “Software requirements” on page 9.)
z/OS Cryptographic Services	Integrated Cryptographic Services Facility with z990 Cryptographic Support Web deliverable (FMID HCR770A) and later releases
Tape or DASD support	Both
Archiving of data	Yes
Exchange of data	Yes
Data portable to other platforms	Yes through Encryption Facility for z/OS Client
zSeries® hardware compression	Yes
zSeries encryption	Yes
Decryption on platforms other than z/OS	Yes through Encryption Facility for z/OS Client

Table 3. (continued)

Support or function	Available with IBM Encryption Facility
zSeries hardware cryptographic acceleration	Yes
Key management through ICSF	Yes (Encryption Services and DFSMSdss Encryption only)
JCL required	Yes (Encryption Services only)
Public key exchange through certificates	Through RACF or ICSF
LDAP integration	No
User interface	JCL control statements, RACF commands, DFSMSdss and DFSMShsm commands for DFSMSdss Encryption, and Encryption Facility for z/OS Client Java code

How Encryption Services and Encryption Facility for z/OS Client work

Encryption Services and Encryption Facility for z/OS Client allow you to encrypt z/OS format data. You can archive the data to tape or DASD and send the data to another platform where you can use either Encryption Services or Encryption Facility for z/OS Client to decrypt the data.

How Encryption Services works

Encryption Services is an optional feature of Encryption Facility that runs on z/OS. It includes the CSDFILEN batch program to encrypt data and the CSDFILDE batch program to decrypt data.

Encryption Services can make use of a symmetric key that is randomly generated to protect the data. It encrypts this data key and stores it with the data in a header record. Encryption Services allows you to protect the data key by using public key architecture, by providing an RSA key label, or, if an RSA public/private key pair is not available, by using a key that is generated from a password that you provide.

CSDFILEN batch program: The input data to be encrypted by CSDFILEN is a sequential file on DASD or tape, PDS or PDSE member, or a z/OS UNIX Systems Services file. You can optionally specify that the input data is to be compressed before it is encrypted.

The output is a sequential file with undefined record format. The output contains information required to decipher the encrypted data. You can specify that the output is to be written to a data set, tape, or DASD. With tape, the block size can be as large as is supported by QSAM (currently, 256 KB) to optimize performance and media space.

You specify the JCL for CSDFILEN as follows:

- Options to control the encryption process on the SYSIN statement
- SYSPRINT DD that contains a statistics report on the data that CSDFILEN encrypts
- SYSUT DD statements for the input data file (SYSUT1) and the output for the encrypted data (SYSUT2)

For information about CSDFILEN, see Chapter 3, “Encrypting files through CSDFILEN of Encryption Services,” on page 15.

CSDFILDE batch program: The input to CSDFILDE is the encrypted data from the CSDFILEN batch program or Encryption Facility for z/OS Client. You can specify that only information about the encrypted data is to be written to SYSPRINT through the JCL statements.

You specify the JCL for CSDFILDE as follows:

- Options to control the decryption process on the SYSIN statement
- SYSPRINT DD that contains a statistics report on the data that CSDFILDE decrypts
- SYSUT DD statements for the input encrypted data (SYSUT1) and the output for the decrypted data (SYSUT2)

For information about CSDFILDE, see Chapter 4, “Decrypting files through CSDFILDE of the Encryption Services,” on page 31.

ICSF callable services: ICSF callable services are invoked for both CSDFILEN or CSDFILDE. For information on which callable services are available to Encryption Facility, see “ICSF callable services for CSDFILEN” on page 30 and “ICSF callable services for CSDFILDE” on page 38.

Statistics report file: The statistics report file contains information about the JCL control statements for the CSDFILEN or CSDFILDE batch jobs, information about the input file that is to be encrypted or decrypted, and performance statistics about the job. For information, see “Format of the statistics report file for CSDFILEN” on page 24 and “Format of the statistics report file for CSDFILDE” on page 34.

How Encryption Facility for z/OS Client works

Encryption Facility for z/OS Client is a Java package that allows you to encrypt and decrypt z/OS format data on non-z/OS platforms. You can transport the encrypted data from Encryption Facility for z/OS Client to other platforms for archiving and also decrypt data that has been encrypted by either Encryption Services or Encryption Facility for z/OS Client.

Java classes for Encryption Facility for z/OS Client provide options like the JCL options for CSDFILEN to encrypt data and CSDFILDE to decrypt data.

You download the Java package for Encryption Facility for z/OS Client from the Web and install the Java reference code on your workstation. For information about obtaining the code, see “Getting started with Encryption Facility for z/OS Client” on page 14.

Figure 1 on page 6 shows how the encryption process works with Encryption Services or Encryption Facility for z/OS Client. You can then archive the encrypted data and decrypt the data through Encryption Services or Encryption Facility for z/OS Client:

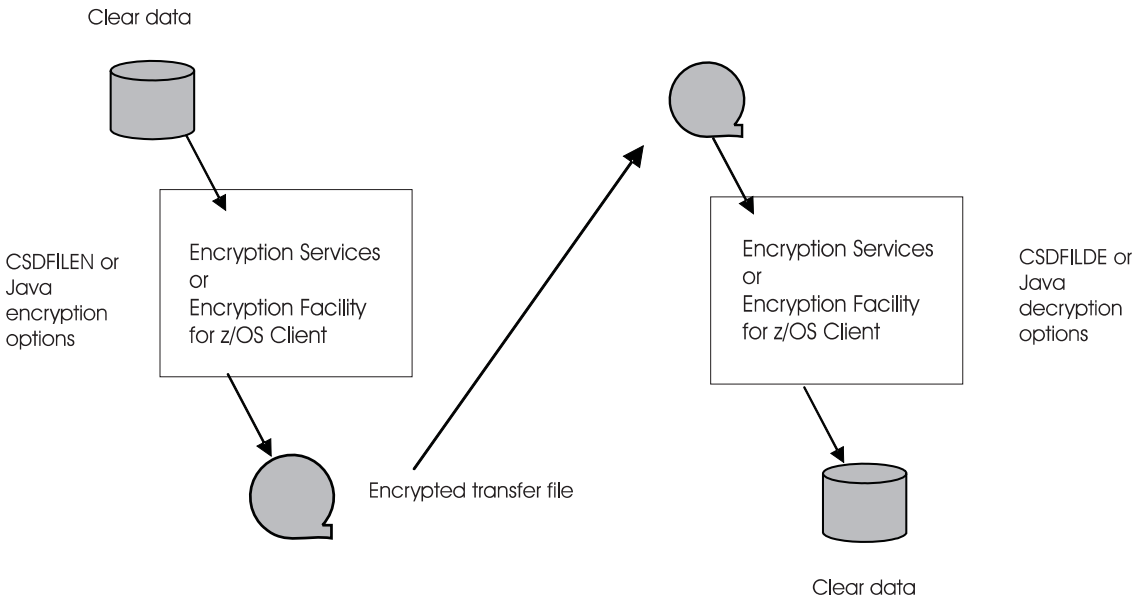

Figure 1. Encrypting and decrypting data with Encryption Services and Encryption Facility for z/OS Client

How DFSMSdss Encryption works

The encryption/decryption process for DFSMSdss Encryption is similar to that of Encryption Services. The DFSMSdss DUMP command and the DFSMShsm DEFINE DUMPCLASS commands provide some of the same options as those for the CSDFILEN batch program of Encryption Services to encrypt data. The DFSMSdss RESTORE command and the DFSMShsm RECOVER commands provide some of the same options as those for the CSDFILEN batch program of Encryption Services to decrypt the data. You can encrypt data to tape or DASD.

See “Software requirements” on page 9.

Figure 2 on page 7 shows how the encryption process works with DFSMSdss Encryption. You use DFSMSdss to encrypt data through the DUMP command where you can archive the data. You can then decrypt the data through the RESTORE command:

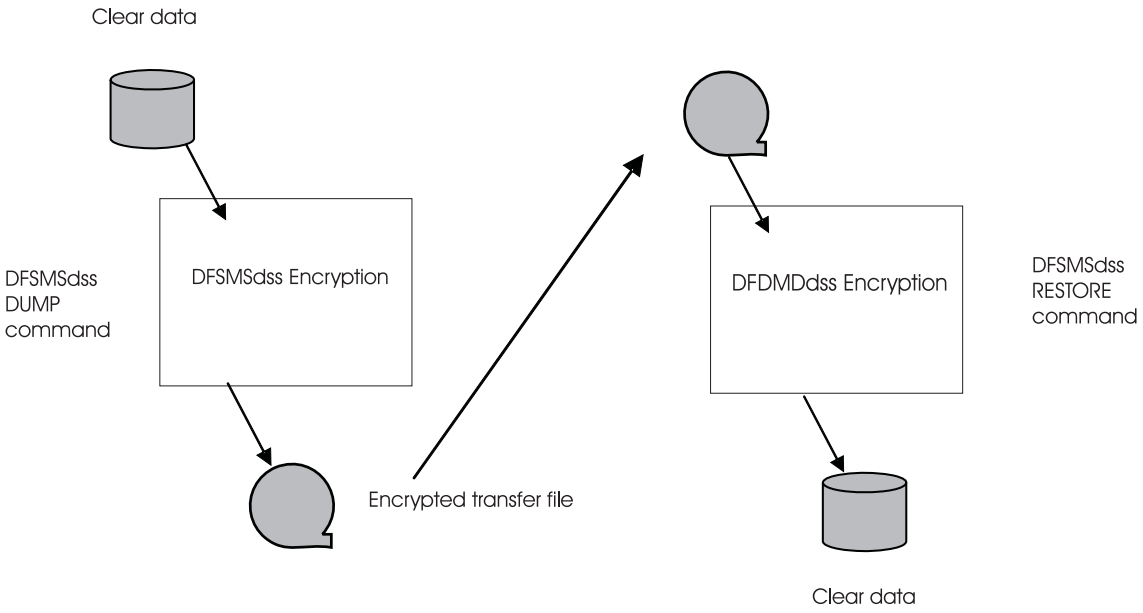

Figure 2. Encrypting and decrypting data with DFSMSdss Encryption

Hardware and software requirements

The following topics describe hardware and software requirements for Encryption Facility.

Hardware requirements

The options that you specify for encryption and decryption including cryptographic keys depend on the processor type and the cryptographic hardware that you have installed on the system. Table 4 summarizes the encryption options, the required processor types, and the required cryptographic hardware for Encryption Facility:

Table 4. Hardware requirements

Encryption options		Processor type	Cryptographic hardware required
Encryption type and key protection options			
• CLRTDES • PASSWORD=	3-key TDES key (24-byte TDES key) generated using password, iteration count, and random salt. Iteration count and random salt values are stored in the header of the encrypted file.	z800 and z900	CCF
		z890 and z990	CPACF
		z9 109	CPACF

Table 4. Hardware requirements (continued)

Encryption options		Processor type	Cryptographic hardware required
Encryption type and key protection options			
• CLRTDES • RSA=	Random 3-key TDES key (24-byte TDES key) generated and encrypted with 512–2048 bit RSA public key. Encrypted key and label of RSA public key are stored in the header of the encrypted file.	z800 and z900	• CCF (for RSA modulus-exponent form keys with modulus length less than or equal to 1024 bits) • PCICC (for RSA Chinese Remainder Theorem keys with modulus length less than or equal to 1024 bits) • CCF and PCICC with LIC January 2005 or later and z990 and z890 Enhancements to Cryptographic Support Web deliverable (ICSF HCR770B) or later for RSA keys with up to 2048 bit modulus
		z890 and z990	PCIXCC or CEX2C
		System z9 109	CEX2C
• CLRAES128 • PASSWORD=	128-bit AES key generated using password, iteration count, and random salt. Iteration count and random salt values are stored in the header of the encrypted file.	z800 and z900	CCF
		z890 and z990	CPACF
		z9 109	CPACF
• CLRAES128 • RSA=	Random 128-bit AES key generated and encrypted with 512–2048-bit RSA public key. Encrypted key and label of RSA public key are stored in the header of the encrypted file.	z800 and z900	• CCF (for RSA modulus-exponent form keys with modulus length less than or equal to 1024 bits) • PCICC (for RSA Chinese Remainder Theorem keys with modulus length less than or equal to 1024 bits) • CCF and PCICC with LIC January 2005 or later and z990 and z890 Enhancements to Cryptographic Support Web deliverable (ICSF HCR770B) or later for RSA keys with up to 2048 bit modulus
		z890 and z990	PCIXCC or CEX2C
		z9 109	CEX2C

Table 4. Hardware requirements (continued)

Encryption options		Processor type	Cryptographic hardware required
Encryption type and key protection options			
• ENCTDES • RSA=	Random 3-key TDES key (24-byte TDES key) generated and encrypted under the systems DES (or symmetric) master key. The TDES key is also encrypted with the 512-2048-bit RSA public key. Encrypted key and label of RSA public key are stored in the header of the encrypted file.	z800 and z900	• CCF (for RSA modulus-exponent form keys with modulus length less than or equal to 1024 bits) • PCICC (for RSA Chinese Remainder Theorem keys with modulus length less than or equal to 1024 bits) • CCF and PCICC with LIC January 2005 or later and z990 and z890 Enhancements to Cryptographic Support Web deliverable (ICSF HCR770B) or later for RSA keys with up to 2048 bit modulus
		z890 and z990	PCIXCC or CEX2C
		z9 109	CEX2C

Software requirements

Software requirements for Encryption Facility are as follows:

Encryption Services and DFSMSdss Encryption: Encryption Services and the DFSMSdss Encryption features of Encryption Facility for z/OS require the following:

- z/OS (5694-A01) or z/OS.e (5655-G52) V1.4 or later release.
- PTF for z/OS DFSMS APAR OA09868 and QSAM APAR OA13571.
- z/OS Cryptographic Services - Integrated Cryptographic Services Facility with z990 Cryptographic Support Web deliverable (FMID HCR770A) or later release. Some hardware features require the z990 and z890 Enhancements to Cryptographic Support Web deliverable (FMID HCR770B or later release). For ICSF levels and FMIDs, see Table 5 on page 10.

RACF (optional): The PTF for APAR OA13030 is required to:

- Specify the PKDS labels to be used when storing public or private keys in the PKDS
- List the PKDS labels of existing certificates
- Use the RACF RACDCERT command to allow the storage of internal RSA public keys in the ICSF PKDS

DFSMSdss Encryption (optional):

DFSMSdss Encryption feature requires:

- The DFSMSdss priced feature of z/OS V1.4 or z/OS.e V1.4 or later release
- PTF for z/OS DFSMS APARs OA13300, OA13453, APAR OA13571, and OA13687

Encryption Facility for z/OS Client (optional):

Encryption Facility Client requires the following:

To run on z/OS, one of the following is required:

- IBM SDK for z/OS, Java 2 Technology Edition, 5655-I56, at PTF UQ90449 or higher (SDK1.4.2)
- IBM Developer Kit for OS/390®, Java 2 Technology Edition, 5655-D35, at PTF UQ88094 or higher (SDK1.3.1)

To run on other platforms, one of the following is required:

- Sun SDK 5.0
- An IBM JVM at SDK1.4.2
- A JVM with a JCE cryptographic provider installed that supports all the required algorithms. See Encryption Facility Client documentation for details on the algorithms, modes, and padding schemes needed.

For complete information, including PTF requirements for iSeries™ or other platforms, see the README file in the Java reference code for Encryption Facility for z/OS Client from the following Web site: http://www.ibm.com/servers/eserver/zseries/zos/downloads/#asis

Table 5 summarizes the ICSF support for Encryption Facility by FMID and z/OS or z/OS.e release:

Table 5. Summary of ICSF support for Encryption Facility

ICSF level	z/OS or z/OS.e release required plus appropriate PTFs	FMID
z990 Cryptographic Support	V1.4 or V1.5	HCR770A
z/OS 1.6 (part of the base)	V1.6	HCR770A
z990 and z890 Enhancements to Cryptographic Support	V1.4 or V1.5	HCR770B
ICSF 64-bit Virtual Support for Z/OS V1.6 and z/OS.e V1.6	V1.6	HCR7720
z/OS 1.7 (part of the base)	V1.7	HCR7720
Cryptographic Support for z/OS V1R6/R7 and z/OS.e V1R6/R7	V1.6 or V1.7	HCR7730

Chapter 2. Getting started

This chapter describes installation tasks and considerations for getting started using IBM Encryption Facility for z/OS:

- "How do I install IBM Encryption Facility for z/OS?"
- "Getting started with Encryption Services"
- "Getting started with ICSF"
- "Getting started with Encryption Facility for z/OS Client" on page 14
- "Getting started with DFSMSdss Encryption" on page 14
- "Getting started with RACF" on page 14

How do I install IBM Encryption Facility for z/OS?

You can install Encryption Facility and use any of the following optional features:

- Encryption Services
- DFSMSdss Encryption

You can download Encryption Facility for z/OS Client from the World Wide Web.

The following steps are a summary of how to install Encryption Facility. These steps provide only a broad description. For installation information, see the *IBM Encryption Facility for z/OS: Program Directory*. For hardware and software requirements for Encryption Facility, see "Hardware and software requirements" on page 7.

1. Ensure that you have the Web Deliverable ″Integrated Cryptographic Services Facility with z990 Cryptographic Support Web deliverable (FMID HCR770A) or later release installed on your z/OS system
2. Order and install program product, 5655-P97 (IBM Encryption Facility for z/OS)
3. Apply any service from PSP Buckets.
4. Optional: Download Encryption Facility for z/OS Client through click through license from the following Web site:http://www.ibm.com/servers/eserver/zseries/zos/downloads/#asis

Getting started with Encryption Services

You can use Encryption Services to encrypt and decrypt data on z/OS. Encryption Services is an SMP/E installable program.

SYS1.SAMPLIB: You can find DDDEF and ALLOC jobs for the Encryption Services in the following SYS1.SAMPLIB members:

- CSDDDDEF for DDDEF
- CSDALLOC for ALLOC

For information about Encryption Services, see Chapter 3, "Encrypting files through CSDFILEN of Encryption Services," on page 15 and Chapter 4, "Decrypting files through CSDFILDE of the Encryption Services," on page 31.

Getting started with ICSF

If you have ICSF installed, see "Software requirements" on page 9 to ensure that you are using the required level for Encryption Facility.

If you need information about installing, planning, and implementing ICSF, see the following publications:

- *z/OS Cryptographic Services ICSF Overview*
- *z/OS Cryptographic Services ICSF System Programmer's Guide*
- *z/OS Cryptographic Services ICSF Administrator's Guide*

Encryption Facility makes use of ICSF to manage cryptographic keys for encrypted data.

ICSF supports the following cryptographic standards and architectures:

- IBM Common Cryptographic Architecture (CCA) that is based on the ANSI Data Encryption Standard (DES)
- Advanced Encryption Standard (AES).

Cryptographic keys: In the secret key cryptography system based on DES, two parties share secret keys that are used to protect data and keys that are exchanged on the network. Sharing secret keys establishes a secure communications channel. The only way to protect the security of the data in a shared secret key cryptographic system is to protect the secrecy of the secret key.

ICSF also supports triple DES encryption for data privacy. TDES triple-length keys use three, single-length keys to encipher and decipher the data. This results in a stronger form of cryptography than that available with single DES encipherment.

With AES, data can be encrypted and decrypted using 128-bit, 192-bit, and 256-bit clear keys. CBC and ECB encryption are also supported.

For public key cryptography, ICSF supports both the Rivest-Shamir-Adelman (RSA) algorithm 1, and the NIST Digital Signature Standard algorithm. RSA is one of the most widely used public key encryption algorithms. In this system, each party establishes a pair of cryptographic keys, which includes a public key and a private key. Both parties publish their public keys in a reliable information source, and maintain their private keys in secure storage.

Cryptographic keys and Encryption Facility: Encryption Facility makes use of TDES triple-length keys and 128–bit AES keys for data encryption. On a system with secure cryptographic hardware, you can use Encryption Facility to generate TDES and AES keys and encrypt them for protection through RSA public keys. On systems without secure cryptographic hardware, a password allows the generation of clear TDES and AES keys. The use of these cryptographic keys with Encryption Facility depends on the kind of processor and the type of cryptographic hardware that you have installed. See Table 4 on page 7.

Generating and placing an RSA key in the PKDS: RSA public and private keys for encryption can be stored in the ICSF public key data set (PKDS). These RSA keys are used by Encryption Facility to protect the symmetric keys that protect the data. You can specify multiple RSA keys as input to Encryption Services or Encryption Facility for z/OS Client and copy and distribute the resulting output file to multiple recipients. You can also use ICSF callable services to generate RSA keys and place them in the PKDS. The required ICSF callable services are CSNDPKB PKA key token build and CSNDPKG PKA key generate.

CSNDPKB builds a skeleton PKA token. The principal parameters are as follows:

- Rule array

- Key Value Structure (KVS)
- Generated Key Token (KeyToken) .

For example, the parameters for the generation of a skeleton key token for a 1024 bit RSA private key are as follows:

- PKB_RULE = ″RSA-PRIVKEY-MGMT″
- PKB_KVS = “040000000030000010001”
- PKB_KeyToken = (generated)

CSNDPKG generates key values for the PKA token. The principal parameters are as follows:

- Rule array
- Skeleton key identifier (SkelKey)
- Generated key identifier (GenKey)

For example, the parameters for a 1024 bit RSA private key are as follows:

- PKG_RULE = ″MASTER″
- PKG_SkelKey = PKB_KeyToken
- PKG_GenKey = “THIS.CAN.BE.A.PKDS.LABEL”

If you specify a PKDS key label for GenKey, ICSF writes the token to the PKDS.

Using the ICSF utility panels to create or delete PKDS records and import or export RSA keys: You can use ICSF utility panels to create or delete PKDS records and export or import RSA keys to an x.509 certificate. You use x.509 certificates to certify the transmission of the RSA public keys between senders and receivers of encrypted data. For information about using digital certificates, see “Using RACF to store keys, manage PKDS labels, and send digital certificates” on page 47.

Coprocessor Requirements for using the ICSF utility panels: To use the full function of the ICSF utility panels, you must have a PCICC, PCIXCC, or a CEX2C cryptographic coprocessor. If you do not have one of these coprocessors, you cannot generate key pairs using the panels.

For information about using the ICSF utility panels, see “Using ICSF utilities panels for PKDS key management” on page 65. For complete information about using ICSF utility panels and services, see *z/OS Cryptographic Services ICSF Administrator’s Guide*.

ICSF uses the following ICSF callable services to create or delete PKDS records and export or import RSA keys to x.509 certificates:

- CSNDKRR (ensures that the specified PKDS label does not already exist)
- CSNDPKB (builds the skeleton key token)
- CSNDKRC (creates the PKDS record)
- CSNDKRD (deletes the PKDS record)
- CSNDKRR (reads the record from the PKDS)
- CSNDPKX (extracts only the public key from the record)
- CSNBOWH (hashes the to-be-signed portion of the generated certificate)
- CSNDDSG (signs the hash)

If you are using RACF or similar security product, ensure that the security administrator authorizes ICSF to use these services and any cryptographic keys

that are input. For information about ICSF callable services, see *z/OS Cryptographic Services ICSF Application Programmer's Guide*.

Getting started with Encryption Facility for z/OS Client

You can use Encryption Facility for z/OS Client to encrypt and decrypt z/OS format data on non-z/OS platforms.

Encryption Facility for z/OS Client is licensed Java reference code that you can download and use with Encryption Facility. The Java program allows you to encrypt and decrypt data that you can transfer between different systems and for archiving purposes.

You can download the Java reference code for Encryption Facility for z/OS Client from the following Web site: http://www.ibm.com/servers/eserver/zseries/zos/downloads/#asis. From this Web site you can also download free decryption services for Encryption Facility for z/OS Client called Decryption Client.

Be sure to read the README file for complete information about software requirements including any PTFs for iSeries or other platforms.

For overview information, see Chapter 5, "Using Encryption Facility for z/OS Client," on page 39.

Getting started with DFSMSdss Encryption

You can use DFSMSdss Encryption of Encryption Facility to perform the encryption of output data sets from the DFSMSdss DUMP command to tape or DASD. You can decrypt the data through the DFSMSdss RESTORE command. With this feature you can also use the DFSMShsm dump class settings to encrypt data dumped through the BACKVOL DUMP command and automatic dump processing. You can use the DFSMShsm RECOVER command to decrypt the data.

See Chapter 6, "Using DFSMSdss Encryption," on page 43.

Getting started with RACF

RACF APAR

For this release of Encryption Facility, you need to ensure that the RACF PTF for APAR OA13030 for z/OS 1.4 or later release systems is installed to be able to use the RACF enhancements for the product.

You can use RACF to help you store RSA public and private keys for encryption in the ICSF public key data set (PKDS). You can also specify the PKDS labels to use when you store public or private keys in the PKDS and can list PKDS labels of public/private key pairs from existing certificates that reside in the RACF database.

The certificate management services of RACF allow you to establish a limited scope certificate authority for your internal and external users, issuing and administering digital certificates in accordance with your own organization's policies.

For information about using RACF to store keys and generate labels, see Chapter 7, "Using RACF with Encryption Facility," on page 47.

Chapter 3. Encrypting files through CSDFILEN of Encryption Services

This chapter presents information about using the CSDFILEN batch program of Encryption Services to encrypt data.

CSDFILEN is the batch program of Encryption Services that encrypts input data files. Depending on the kind of processor and cryptographic coprocessor, you can specify options to use clear key or secure key encryption, specify options for key protection, and indicate if you want to compress the data before it is to be encrypted.

JCL DD statements for CSDFILEN

CSDFILEN supports the following DD statements or their dynamic allocation equivalents:

Table 6.

DD statement	Description
SYSPRINT	Specifies the name of the data set to which CSDFILEN writes encryption statistics and diagnostics information. It can be a sequential data set as follows: • Typically a sysout data set • DASD or tape data set • PDS or PDSE member CSDFILEN sets the following values: • RECFM=FBA • LRECL=133 The system selects an optimal value for BLKSIZE unless you choose to code BLKSIZE. BLKSIZE must be a multiple of 133 unless it is a sysout DD statement, but coding BLKSIZE on a sysout DD statement is not beneficial.
SYSIN	Specifies the source from which CSDFILEN reads control statements. It can be a sequential data set as follows: • Typically a spooled system input data set • ONE of the following specifications: – DASD or tape data set – PDS or PDSE member CSDFILEN requires the following specifications: • RECFM=F or FB • LRECL=80

Table 6. (continued)

DD statement	Description
SYSUT1	Specifies the name of the data set that contains data to be encrypted. It can be a sequential data set as follows: • DASD or tape data set • PDS or PDSE member • z/OS UNIX Systems Services file The input to CSDFILEN can also be a system input data set, that is if you code * or DATA on the DD statement. Unlike the other types of input, a dynamic allocation equivalent of this data set does not exist. CSDFILEN can read data sets created with BSAM, QSAM, BPAM or EXCP. You must first unload VSAM data in order to encrypt it. For z/OS UNIX Systems Services files, consider the following coding practices: • Code the PATH keyword and PATHOPTS=ORDONLY. • Do not code FILEDATA=TEXT so that CSDFILEN can read the data without character conversion. • Do not code variable block (VB) for BLKSIZE. For all types of input consider the following for record format and DCB information: • The data set or file can have any record format (RECFM). • If the data set label (whether on DASD or tape) contains the DCB information, you do not have to code RECFM, LRECL or BLKSIZE on the DD statement. • If you plan to decrypt the file to a z/OS UNIX Systems Services file or send the file to a non-z/OS system, code RECFM=U so that you get the maximum length of records for better efficiency. Normally DASD data sets and standard labelled tapes have the correct DCB information. If the record format is not available in the data set label and you do not code RECFM on the DD statement, CSDFILEN assumes RECFM=U (undefined format). If the block size is not available from the data set label and you do not code BLKSIZE on the DD statement, CSDFILEN assumes the maximum block size supported by the device. If this block size value is much larger than the sizes of the real blocks, the program might run more slowly than you expect. IBM suggests that if the input is an unlabeled dump tape created by DFSMSdss, you code BLKSIZE=60000, which is a little larger than the largest block expected. If CSDFILEN is using a record format of fixed or variable and the record length is not available from the data set label and you do not code LRECL on the DD statement, CSDFILEN fails.

Table 6. (continued)

DD statement	Description
SYSUT2	Specifies the name of the data set that is to contain the encrypted data. It can be a sequential data set as follows: • DASD or tape data set • PDS or PDSE member CSDFILEN forces RECFM=U. You can specify the maximum block size by coding the BLKSIZE or BLKSZLIM keyword. With either keyword, if the value exceeds the maximum supported by the device, CSDFILEN reduces the value. If you do not code a value, CSDFILEN assumes the optimal value for the device. For information about these values, see the INFO=AMCAP option of the DEVTYPE macro in *z/OS DFSMSdfp Advanced Services*. If you want to copy the file containing the encrypted data to another device (for example, you specify a tape data set on SYSUT2 but later copy the data set to DASD), you might want to code BLKSIZE to reflect a block size that is applicable to the other device (for example, 23476 for DASD). The following list is a summary of default block sizes depending on the kind of specified device: • If the device is DASD, the default block size is a half track. • If the device is an IBM 3590 or a newer tape device, the default block size is 256 KB. • If the device is an IBM 3480 or 3490 Tape Subsystem or an IBM Virtual Tape Server, the default block size is 65535. Note that this data set cannot be an ANSI/ISO standard labelled tape or a tape for which OPTCD=Q is specified. In either case, the system requires the data to be character data and performs character conversion, which thereby destroys encrypted data. **DO NOT specify DISP=MOD for the SYSUT2 data set on CSDFILEN.** You might encounter an error when you try to decrypt the data through CSDFILDE.

Control statement keywords for CSDFILEN SYSIN DD

You can specify the following options in the control statement data set (identified by SYSIN DD) to control encryption of the input files. All keywords must start in column 1, and you cannot code a continuation statement. The program treats an asterisk (*) in column 1 as a comment. If you specify the same keyword multiple times, the program uses the last specification:

Table 7.

Description	JCL keyword
Descriptive text	**DESC='***text***'** Specifies 1 to 64– EBCDIC character bytes of descriptive *text* to be placed in the header record. CSDFILEN places the information in the header record. The information is used to assist in identifying the source of the encrypted data in the output. You must enclose the text in single quotation marks. Imbedded blanks are allowed. All text must be included on one control statement line. DESC is optional.

Table 7. (continued)

Description	JCL keyword
Encryption type	Specifies information about which encryption key you want Encryption Services to generate. You can specify one of the following types. If you do not specify an option, the default is CLRTDES: **CLRTDES** Specifies that the input file is to be encrypted with a clear TDES triple-length key. **CLRAES128** Specifies that the input file is to be encrypted with a clear 128-bit AES key. **ENCTDES** Specifies that the input file is to be encrypted with a secure TDES triple-length key.
Method to generate and protect the data encrypting key	Specifies the method to be used to generate and protect the data encrypting key. RSA and PASSWORD are mutually exclusive. One of the following keywords is required: **RSA=***label* Specifies the 64-byte *label* of an existing RSA public key that is in the ICSF PKDS. The program uses this key to encrypt the data-encrypting key. The corresponding RSA private key must be present at the recovery site when you decipher the data. RSA= can point to an RSA key that contains both a private and a public key, or you can specify the name of the corresponding RSA private key when you invoke CSDFILDE or Encryption Facility for z/OS Client at the recovery site. For Encryption Services you can use from 1 to 16 RSA= keywords to specify from 1 to 16 public key labels. Depending on the number of multiple RSA labels, you can send the encrypted file to up to 16 individual recipients. For how to specify multiple RSA key labels, see “Specifying multiple RSA keys” on page 21. **PASSWORD=***password* Specifies the 8- to 32- EBCDIC character *password* to be used to generate a clear TDES triple-length key or a clear 128-bit AES key. Leading and trailing blanks and tab characters are removed; imbedded blanks and tab characters are allowed. Passwords are case sensitive. You can specify this option on systems that do not have secure cryptographic hardware (for example, for z890, z990 or z9-109 processors that only use CPACF). In order to minimize problems because of code page differences at the encrypting and decrypting sites, IBM suggests that you use only the upper and lower-case letters A through Z, numerals 0 – 9 and the underscore character (_).

Table 7. (continued)

Description	JCL keyword
Number of iterations	**ICOUNT=***nnnnn* When you specify a password, specifies the number of iterations that the SHA-1 hash algorithm is to be performed in the generation of the data key and the initial chaining vector (ICV) for encryption. nnnnn is an integer between 1 and 10000. If you do not specify ICOUNT, the default is 16. ICOUNT allows you to strengthen security when you use PASSWORD. If you specify a robust password (32 random characters), the default is sufficient. See “Specifying the ICOUNT value” on page 21.
Compression option	**COMPRESSION=NO|YES** Specifies whether you want compression of the clear input before encryption of the data occurs. COMPRESSION=NO indicates that compression does not occur. COMPRESSION=YES causes compression to be performed before encryption. If you do not specify the COMPRESSION keyword, the default is NO.

User guidelines and samples for encrypting data

Guidance information includes the following topics:

Reference information includes the following topics and samples:

When should I use CLRTDES or ENCTDES?

The decision on whether to use CLRTDES or ENCTDES key values depends on the kind of cryptographic hardware you have, the level of security you want, and the level of performance.

For Encryption Facility, a CLRTDES key is a triple-length TDES key that the service generates dynamically. Unlike the ENCTDES key value the CLRTDES key value can appear in application storage. If Encryption Facility is running on a z890, z990, or System z9 109, CSDFILEN encrypts the data using the clear TDES key on the CPACF. This usually results in better performance than if you are using the ENCTDES key value.

The ENCTDES key is a triple-length TDES key that the service generates within the secure boundary of the cryptographic hardware (CCF, PCICC, PCIXCC, or CEX2C), and it uses the ICSF symmetric master key to encrypt the data. The clear value of an ENCTDES key never leaves the boundary of the secure cryptographic hardware. Encryption and decryption of data using an ENCTDES key requires secure cryptographic hardware to be available.

Each type of key is equally secure in regards to the data that appears in the output file of the CSDFILEN statistics report file; that is, CSDFILEN does not write any clear key information to the file.

Using PASSWORD and RSA options

PASSWORD and RSA options for the encrypted data depend on the processor and cryptographic hardware that you have installed. Base your decisions on the security requirements of the data and on your available hardware.

PASSWORD option: Generally, if you do not have secure cryptographic hardware installed, you can specify the PASSWORD keyword. Passwords are case sensitive.

RSA option: Consider using the RSA keyword that makes use of public/private keys for encryption and the exchange of digital certificates. You specify the label of the public key that is stored in the ICSF PKDS on the RSA option when you encrypt the data. The corresponding RSA private key must be present at the recovery site when you decipher the data. A recipient on another site can decrypt the data through the private key that is specified on the RSA option for CSDFILDE. Optionally, the recipient can decrypt the data through the private key that is specified by the **–keyStoreCertificateAlias** argument of the Encryption Facility for z/OS Client; see “Using RSA keys and certificates” on page 39. You can specify multiple RSA keys as input.

See “Control statement keywords for CSDFILDE SYSIN DD” on page 33.

RSA private tokens: When you use the RSA option with CSDFILEN to encrypt the data-encrypting key, you must consider the cryptographic hardware that exists at the site that decrypts the data. All types of RSA private keys are not supported by all types of cryptographic hardware. Table 8 summarizes the RSA private tokens and required cryptographic hardware for decryption. For details, see *z/OS Cryptographic Services ICSF Application Programmer's Guide*:

Table 8. RSA private tokens and required cryptographic hardware

RSA private key token (internal)	Required cryptographic hardware
RSA private key token 1024 Modulus-Exponent Internal form	One of the following: • Cryptographic Coprocessor Feature • PCI X Cryptographic Coprocessor • Crypto Express2 Coprocessor
RSA private key token 1024 Chinese Remainder Theorem Internal form	One of the following: • PCI Cryptographic Coprocessor • PCI X Cryptographic Coprocessor • Crypto Express2 Coprocessor

Table 8. RSA private tokens and required cryptographic hardware (continued)

RSA private key token (internal)	Required cryptographic hardware
RSA private key token 2048 Chinese Remainder Theorem Internal form	One of the following: • PCI Cryptographic Coprocessor with LIC January 2005 or later and z/OS ICSF HCR770B or later • PCI X Cryptographic Coprocessor • Crypto Express2 Coprocessor

Specifying multiple RSA keys

You can specify multiple RSA control statements. Each control statement identifies the label of one RSA public key in the ICSF PKDS. CSDFILEN supports up to 16 public key labels.

For each RSA public key, Encryption Services creates an identifier that is to be associated with the key. The RSA identifier allows CSDFILDE to associate an RSA private key with the correct RSA key information in the header record of the encrypted output file. To decrypt the file that uses the RSA option, you must specify the **RSA=** control statement with the label of the RSA private key on CSDFILDE or specify the **-keyStoreCertificateAlias** parameter for the RSA private key on Encryption Facility for z/OS Client. CSDFILDE supports only one **RSA=** control statement. The statistics report file for CSDFILEN includes any **RSA=** control statements that you specify as input.

Using RSA keys and digital certificates

When you use RSA for cryptographic key management instead of PASSWORD (derived key option), digital certificates form the basis of the key exchange. The ″public keys″ that are stored in the ICSF PKDS are almost always derived from digital certificates. You can use your own program to store public and private RSA keys and manage certificates, but if you use RACF, the RACDCERT command allows you to perform the following functions:

- Store internal public/private RSA keys in the ICSF PKDS
- Manage PKDS labels for the keys
- Establish a limited scope certificate authority for your users

See Chapter 7, “Using RACF with Encryption Facility,” on page 47.

Specifying the ICOUNT value

The iteration count (ICOUNT) in Password Based Encryption (PBE) is intended to strengthen weak passwords. The default of 16 provides reasonable security and performance if the password is robust (that is, 32 random characters). Most PBE schemes assume that you choose a weak password; thus, iteration counts of 1000 or higher are often normal.

When should I compress data for encryption?

When you plan to archive large amounts of encrypted data, you might consider compressing the data (for example, to reduce the number of tape volumes you might need).

Some tape devices make use of their own compression when you store data. Encrypted data is not highly compressible, so you might want to compress your data before encryption using the COMPRESSION option of Encryption Services.

If you plan to use Encryption Facility for z/OS Client to decrypt z/OS data, note that it cannot decrypt data that has been compressed from the CSDFILEN batch program.

Encryption Facility is able to compress 64 K bytes or more of data. If you try to compress less than that amount, the statistics report output for CSDFILEN indicates that 0 bytes of data have been compressed. If you try to decompress the data through CSDFILDE, the statistics report output for CSDFILDE indicates that 0 bytes have been expanded.

Verifying encryption files when you archive

Before you send any files that you encrypt to other systems for archiving, verify that you can decrypt the file on the same system where you encrypted it. Also, be sure to retain your keys for encrypted data especially if you plan to archive the data for a long time. See Chapter 4, “Decrypting files through CSDFILDE of the Encryption Services,” on page 31.

Using Encryption Facility and UNIX pipes

You can utilize a UNIX pipe to improve performance of Encryption Services if you have a utility that writes data to a data set and the data set is to be input to Encryption Services. Instead of using an intermediate file or data set, you can make use of a UNIX pipe to pipe the data directly to the Encryption Services. As a result, you might obtain significant performance improvements for UNIX Systems Services applications.

UNIX pipes are supported wherever you use either a fixed or undefined record format file for the pipe data set, as long as the JCL provides appropriate values for LRECL, BLKSIZE, and RECFM on the DD statement. For example, the following DD statement defines a fixed record format for a UNIX pipe that another application has created and opened for writing:

```
//SYSUT1   DD   PATH='/tmp/temppipe',
//           LRECL=4160,BLKSIZE=4160,RECFM=FB,
//           DSNTYPE=PIPE,
//           FILEDATA=BINARY,
//           PATHOPTS=(ORDONLY),
//           PATHMODE=SIRWXU
```

For complete information about UNIX pipes, see *z/OS UNIX System Services User's Guide*.

Format of the header record for the CSDFILEN output file

The output of the CSDFILEN program contains the encrypted data with a header record that contains the information you need for the CSDFILDE program or Encryption Facility for z/OS Client to decrypt the data. Table 9 shows the format of the header record:

Table 9.

Offset (Decimal)	Name of Header field	Type of data	Description
0	HEADER_EYE	Character	Eyecatcher: ″HEADER″ in EBCDIC.

Table 9. (continued)

Offset (Decimal)	Name of Header field	Type of data	Description
6	HDR_VERSION	Character	Version of the header record for Encryption Facility.
8	HDR_DESC	Character	EBCDIC description (DESC keyword) of CSDFILEN input file.
72	HDRLEN	Integer	Length of entire header record (integer format).
76	HDRSALT	Character	8-byte field (salt value) used with password.
84	HDRICNT	Integer	Iteration count (ICOUNT keyword), integer format from 1 - 10000 to be used with password.
88	HDRKEYLN	Character	Modulus length (hexadecimal format from 512 – 2048) in bits of the RSA public/private key taken from the RSA keyword information.
90	HDRRSA	Character	64-byte label (RSA keyword) of the RSA public/private key in ICSF PKDS.
154	HDRICV	Character	Initialization chaining vector to be used with encryption/decryption.
170			Reserved for IBM use.
174	HDAESDES	Bit	Type of key to be used to encrypt/decrypt data: • x'01' use a clear TDES triple–length key • x'02' use a clear 128-bit AES key • x'03' use a secure TDES triple–length key .
175	HDRFLAGS	Bit	Bit string that indicates type of output, compression options, and format of encrypted data: • Bit 0 = '1'b: unused • Bit 1 = '1'b: indicates output data compressed • Bit 2 = '1' b: indicates compression dictionary is present in the encrypted data • Bit 3 = '1'b: indicates clear data is binary • Bit 3 = '0'b: indicates clear data is text (not used by z/OS) .
176	HDR_COMPVER	Character	Version of Encryption Facility compression used.
178	HDRIRECF	Bit	Input file record format: • Bit 0 = '1'b, Bit 1 = '0'b: Fixed • Bit 0 = '0'b, Bit 1 = '1'b: Variable • Bit 0 = '1'b, Bit 1 = '1'b: Undefined • Bit 3 = '1'b: Blocked records • Bit 5 = '1'b: ASA control character • Bit 6 = '1'b: Machine control character.
179	HDRIRECL	Integer	Input file logical record length
181	HDRIBLKS	Integer	Input file block size
185	HDRORECF	Bit	Output file record format: • Bit 0 = '1'b, Bit 1 = '0'b: Fixed • Bit 0 = '0'b, Bit 1 = '1'b: Variable • Bit 0 = '1'b, Bit 1 = '1'b: Undefined • Bit 3 = '1'b: Blocked records • Bit 5 = '1'b: ASA control character • Bit 6 = '1'b: Machine control character.
186	HDRORECL	Integer	Length of output file logical record.

Table 9. (continued)

Offset (Decimal)	Name of Header field	Type of data	Description
188	HDROBLKS	Integer	Output file block size.
192	HDR_KEYVAL	Integer	Encrypted data-encrypting key.
448			Reserved for IBM use.
464	HDR_RSA_CNT	Integer	**Applies only when the ″HEADER″ is version X'0002' or greater:** Number of RSA= control statements.
468	HDR_RSA	Character	**Applies only when the ″HEADER″ is version X'0002' or greater:** An array consisting of information for multiple RSA= control statements. The length is variable based on the number of RSA= control statements with each entry 344 bytes in length.
	HDR_RSA_LAB	Character	An element of HDR_RSA consisting of a 64-byte label of one of the RSA public/private keys in ICSF PKDS.
532	HDR_KEY_LN	Character	An element of HDR_RSA consisting of Modulus length (hexadecimal format from 512 - 2048) in bits of the RSA public/private key in this entry.
534		Character	Two-byte placeholder of HDR_RSA.
536	HDR_KEY_VAL	Character	An element of HDR_RSA consisting of the hexadecimal encrypted data-encrypting key. This value is encrypted by the RSA key in this entry.
792	HDR_RSA_TAG	Character	An element of HDR_RSA consisting of a hexadecimal value used for validation.
812			End of Header record.

Format of the statistics report file for CSDFILEN

The output of the statistics report file depends on whether the encrypted data has been compressed.

COMPRESSION=NO: The following example shows the output of the statistics report file from CSDFILEN when compression is not specified (COMPRESSION=NO):

```
    CSDFILEN Encryption Utility 09/28/2005 (MM/DD/YYYY) 12:43:38 (HH:MM:SS)
INPUT:  DESC='DATA TO SEND TO PARTNER'
INPUT:  RSA=CCA.PVT06.INT.ENC.1024S0F
INPUT:  COMPRESSION=NO
 CSDFILEN:    RSA-PUB : CCA.PVT06.INT.ENC.1024S0F
INPUT:  LRECL   121 BLKSIZE     484 RECFM FB
OUTPUT: BLKSIZE   32760
ENCRYPTION OF DATA: CLEAR     TDES KEY USING CPACF
  RECORDS READ:         22,653  WRITTEN:             88
  BYTES READ:          2,741,013
  BYTES WRITTEN:         2,877,408  WITH HEADER AND PAD
CIPHER TIMES (IN SECONDS): HIGH:    0.001969 DATA:   306704 LOW:    0.000747 DATA: 116600
 TOTAL CIPHER TIME (IN SECONDS):     0.018274  CIPHERS:             10
  TOTAL ELAPSED TIME:   0:00:04.12
```

COMPRESSION=YES: The following example shows the output of the statistics report file from CSDFILEN when compression is specified (COMPRESSION=YES):

```
        CSDFILEN Encryption Utility 09/28/2005 (MM/DD/YYYY) 12:51:20 (HH:MM:SS)
INPUT:  DESC='COMPRESSED DATA FOR FILE XYZ'
INPUT:  CLRAES128
INPUT:  PASSWORD=********************************
INPUT:  COMPRESSION=YES
 CSDFILEN:            :
INPUT:  LRECL   121 BLKSIZE     484 RECFM FB
OUTPUT: BLKSIZE   32760
ENCRYPTION OF DATA: CLEAR     AES  KEY USING CSNBSYE
  RECORDS READ:         22,653  WRITTEN:             25
  BYTES READ:          2,741,013
  BYTES WRITTEN:           789,376  WITH HEADER AND PAD
CIPHER TIMES (IN SECONDS): HIGH:    0.003655 DATA:   123376 LOW:    0.000880 DATA:  29968
 TOTAL CIPHER TIME (IN SECONDS):     0.022709  CIPHERS:             10
  TOTAL COMPRESS TIME (IN SECONDS):     0.232279 TOTAL COMPRESS BYTES:        723,320
  TOTAL ELAPSED TIME:   0:00:02.43
```

If you try to compress less than 64 K of data, CSDFILEN does not compress the data, and the statistics report indicates the following for COMPRESS BYTES:

```
COMPRESS BYTES:     0
```

See “When should I compress data for encryption?” on page 21.

Multiple RSA control statements: The following example shows the output of the statistics report file from CSDFILEN that includes the maximum of 16 RSA label definitions:

```
         CSDFILEN Encryption Utility 02/15/2006 (MM/DD/YYYY) 18:21:
INPUT:  *-------------------------------------------------------
INPUT:  *    ENC4: Multiple RSA + CLRTDES Encryption + Compression
INPUT:  *-------------------------------------------------------
INPUT:  DESC='Multiple RSA + CLRTDES Encryption + Compression'
INPUT:  RSA=RSA.ME02.512.PRIV.KEYMGMT.CLEAR
INPUT:  RSA=RSA.514.PUBLIC
INPUT:  RSA=RSA.799.INTERNAL.PRIVATE.TOKEN
INPUT:  RSA=RSA.ME06.513.PRIV.KMONLY
INPUT:  COMPRESSION=YES
INPUT:  RSA=RSA.ME06.651.PRIV.KEYMGMT
INPUT:  RSA=RSA.ME06.1023.PRIV.KEYMGMT
INPUT:  RSA=RSA.ME06.1024.PRIV.KMONLY
INPUT:  RSA=RSA.CRT08.512.PUB.KEYMGMT.CLEAR
INPUT:  RSA=RSA.CRT08.675.PRIV.KMONLY.CLEAR
INPUT:  CLRTDES
INPUT:  RSA=RSA.CRT08.1024.PRIV.KEYMGMT
INPUT:  RSA=RSA.CRT08.1243.PRIV.KMONLY.CLEAR
INPUT:  RSA=RSA.CRT08.1666.PRIV.KEYMGMT.CLEAR
INPUT:  RSA=RSA.CRT08.2048.PUB.KEYMGMT.CLEAR
INPUT:  RSA=RSA.CRT08.PRIV.1536.BIT.MODULUS
INPUT:  RSA=RSA.ME06.PRIV.830.BIT.MODULUS
INPUT:  RSA=RSA.ME06.PRIV.833.BIT.MODULUS
 CSDFILEN:    RSA-PUB : RSA.ME02.512.PRIV.KEYMGMT.CLEAR
 CSDFILEN:    RSA-PUB : RSA.514.PUBLIC
 CSDFILEN:    RSA-PUB : RSA.799.INTERNAL.PRIVATE.TOKEN
 CSDFILEN:    RSA-PUB : RSA.ME06.513.PRIV.KMONLY
 CSDFILEN:    RSA-PUB : RSA.ME06.651.PRIV.KEYMGMT
 CSDFILEN:    RSA-PUB : RSA.ME06.1023.PRIV.KEYMGMT
 CSDFILEN:    RSA-PUB : RSA.ME06.1024.PRIV.KMONLY
 CSDFILEN:    RSA-PUB : RSA.CRT08.512.PUB.KEYMGMT.CLEAR
 CSDFILEN:    RSA-PUB : RSA.CRT08.675.PRIV.KMONLY.CLEAR
 CSDFILEN:    RSA-PUB : RSA.CRT08.1024.PRIV.KEYMGMT
 CSDFILEN:    RSA-PUB : RSA.CRT08.1243.PRIV.KMONLY.CLEAR
 CSDFILEN:    RSA-PUB : RSA.CRT08.1666.PRIV.KEYMGMT.CLEAR
 CSDFILEN:    RSA-PUB : RSA.CRT08.2048.PUB.KEYMGMT.CLEAR
 CSDFILEN:    RSA-PUB : RSA.CRT08.PRIV.1536.BIT.MODULUS
 CSDFILEN:    RSA-PUB : RSA.ME06.PRIV.830.BIT.MODULUS
 CSDFILEN:    RSA-PUB : RSA.ME06.PRIV.833.BIT.MODULUS
INPUT:  LRECL   252 BLKSIZE   32760 RECFM FB
OUTPUT: BLKSIZE   32760
ENCRYPTION OF DATA: CLEAR     TDES KEY USING CCF
  RECORDS READ:            679  WRITTEN:              4
  BYTES READ:          171,108
  BYTES WRITTEN:           100,564  WITH HEADER AND PAD
CIPHER TIMES (IN SECONDS): HIGH:    0.000164 DATA:    94592 LOW:    0.000164 DATA:    94592
 TOTAL CIPHER TIME (IN SECONDS):     0.000164  CIPHERS:             1
  TOTAL COMPRESS TIME (IN SECONDS):     0.092435 TOTAL COMPRESS BYTES:         29,047
  TOTAL ELAPSED TIME:   0:00:20.82
```

Understanding the statistics report

In the statistics report, the line that starts `CIPHER TIMES (IN SECONDS):` shows the longest time (HIGH) that the program takes to encipher a chunk of data and the number of bytes of clear data that are in that chunk. `CIPHER TIMES (IN SECONDS)` also shows the shortest time (LOW) that the program takes to encipher a chunk of data, and the number of bytes in that chunk. For example if the statistics report contains the following line:

```
CIPHER TIMES (IN SECONDS): HIGH:    0.003655 DATA:   123376 LOW:    0.000880 DATA:  29968
```

the longest amount of time taken for a single encipher is .003655 seconds to encipher a block of 123376 bytes. The shortest amount of time taken for a single encipher is .000880 seconds to encipher a block of 29968 bytes.

CSDFILEN diagnostics

CSDFILEN might write diagnostic information to the statistics report file.

The following error line begins with the characters **ERROR** and indicates that more than 16 **RSA=** keywords are specified. CSDFILEN discontinues processing:

```
 RSA SPECIFIED MORE TIMES THAN THE ALLOWED MAXIMUM
```

Each of the following error lines begins with the characters **ERROR** and ends the processing of CSDFILEN:

```
DISP=MOD NOT ALLOWED
DESCRIPTION TEXT MUST BE ENCLOSED IN SINGLE QUOTES
SPECIFY ONE OF RSA/PASSWORD ONLY
ONE OF RSA/PASSWORD MUST BE SPECIFIED
ENCTDES NOT VALID WITH PASSWORD
ITERATION COUNT NOT VALID
UNEXPECTED ERROR DURING COMPRESSION (RETCODE=nn)
SYSUT1 FILE FAILED TO OPEN
SYSUT2 FILE FAILED TO OPEN
MULTIPLE ENCRYPTION TYPES SPECIFIED
PASSWORD ENTERED NOT VALID
RECORD FORMAT FOR SYSUT2 NOT VALID. RECFM(U) IS REQUIRED.
CONFLICTING LRECL, BLKSIZE, AND RECFM
PRODUCT DEREGISTRATION FAILED
```

CSDFILEN might also write the following diagnostic information (beginning with the characters **INFO**) to the statistics report file:

```
CSNBOWH possible SAF authority violation
```

Compression warnings and errors: The following error lines occur together and begin with the characters **WARNING**. They indicate that because of the randomness of the input data, compression does NOT reduce the size of the output file. This is a minor error. CSDFILEN disables compression, but encryption of the data continues:

```
**WARNING** MINOR ERROR OCCURRED BUILDING THE COMPRESSION DICTIONARY.
**WARNING** ENCRYPTION CONTINUING WITHOUT COMPRESSION.
```

The following error lines occur together and begin with the characters **ERROR**. They indicate that an unrecoverable error occurred during compression. This is a major error, and CSFDFILEN cannot continue processing. Turn off compression and rerun the job:

```
**ERROR** CATASTROPHIC ERROR WHILE BUILDING THE COMPRESSION DICTIONARY.
**ERROR** PROCESSING HALTED, RERUN WITHOUT COMPRESSION.
```

ICSF callable services and diagnostics: If CSDFILEN invokes an ICSF callable service and that service encounters a failure, CSDFILEN writes the following diagnostic information to the statistics report file. In this example, the service CSNDSYI was invoked and returned a return code of 8 and a reason code of X'271C'. For information about ICSF return codes and reason codes, see *z/OS Cryptographic Services ICSF Application Programmer's Guide*.

```
**ERROR**CSNDSYI   08  00271C
```

Return codes for CSDFILEN

CSDFILEN can issue the following return codes (decimal values in general register 15):

Table 10.

Return code	Meaning
0	Successful operation
8	User error
20	Physical error occurred on input file
24	Physical error occurred on output file
28	SYSPRINT file failed to open
32	SYSUT1 file failed to open
36	SYSUT2 file failed to open
50	Product registration/deregistration failed

CSDFILEN can also return the following return codes that the ICSF callable service passes back to the routine. In this case, the return code and reason codes from the ICSF service are also recorded in the statistics report file:

Table 11.

Return code	Meaning
4	Warning
8	Application error
12	CSF error
16	Terminating error

JCL Examples for CSDFILEN

Example 1: In this example CSDFILEN reads data from a DASD data set named LAB.MASTER DATA. It creates an encrypted version in a DASD data set named LAB.MASTER.DATA.SAFE. The SYSIN options are for a z800 or z900 processor with CCF or a z890 or z990 with PCIXCC or CEX2C, or z9 109 processor with CEX2C. The options include a description for the encrypted data, a request to generate a secure triple-length TDES key (ENCTDES) protected in the header with a public RSA key:

```
//ENCRYPT1 EXEC PGM=CSDFILEN
//SYSPRINT DD SYSOUT=*
//SYSUT1   DD DSN=LAB.MASTER.DATA,DISP=SHR
//SYSUT2   DD DSN=LAB.MASTER.DATA.SAFE,
//         UNIT=SYSDA,DISP=(NEW,CATLG),
//         SPACE=(1024,(60,10)),AVGREC=K
//SYSIN    DD *
DESC='My Secure Data'
ENCTDES
RSA=ICSFEHN.RSAPUB
/*
//
```

Example 2: In this example CSDFILEN reads data from a DASD data set named LAB.MASTER DATA. It creates an encrypted version in a DASD data set named LAB.MASTER.DATA.SAFE. The SYSIN options are for a z800 or z900 processor with CCF or a z890 or z990 with PCIXCC or CEX2C, or z9 109 processor with CEX2C. The options include a description for the encrypted data, a request to generate a secure triple-length TDES key (ENCTDES) protected in the header with multiple (5) RSA keys:

```
//ENCRYPT1 EXEC PGM=CSDFILEN
//SYSPRINT DD SYSOUT=*
//SYSUT1 DD DSN=LAB.MASTER.DATA,DISP=SHR
//SYSUT2 DD DSN=LAB.MASTER.DATA.SAFE,
// UNIT=SYSDA,DISP=(NEW,CATLG),
// SPACE=(1024,(60,10)),AVGREC=K
//SYSIN DD *
DESC='My Secure Data'
ENCTDES
RSA=ICSFEHN.RSAPUB
RSA=RSA.ME02.512.PRIV.KEYMGMT.CLEAR
RSA=RSA.514.PUBLIC
RSA=RSA.799.INTERNAL.PRIVATE.TOKEN
RSA=RSA.ME06.513.PRIV.KMONLY
/*
//
```

Example 3: In this example CSDFILEN reads data from an IBM standard labeled tape data set named ADRDSSU.DUMPFILE, which is a cataloged data set. In the case of multiple volumes, CSDFILEN allocates two drives to improve performance. It creates an encrypted version on another tape whose volume serial is ARCHIV. The system adds the name of the resulting data set to the system catalog. The SYSIN options are for a z800 or z900 processor with CCF or a z800, z900, or z9 109 with CPACF. The options include a description for the encrypted data, a request to generate a clear triple-length TDES key with a password. An iteration count is also specified and the data is to be compressed.

```
//ENCRYPT2 EXEC PGM=CSDFILEN
//SYSPRINT DD SYSOUT=*
//SYSUT1   DD UNIT=(,2),DSN=ADRDSSU.DUMPFILE,DISP=OLD
//SYSUT2   DD UNIT=3590-1,DISP=(,CATLG),VOL=SER=ARCHIV,
//         DSN=FILE.ARCHIVE
//SYSIN    DD *
DESC='My Secure Data'
CLRTDES
PASSWORD=1509TY6E
ICOUNT=3200
COMPRESSION=YES
/*
//
```

Example 4: In this example CSDFILEN reads data from a z/OS UNIX Systems Services file named `/u/Lab/experiments/test3`. It creates an encrypted version on scratch tapes to be retained. The output data set name is not to be added to the system catalog. The volume serial number for the data set is indicated in the messages for the job. The SYSIN options are for a z800 or z900 with CCF or a z890, z990, or z9 109 processor with CPACF. For hardware encryption, 128-bit AES must be enabled on the processor; otherwise, software encryption occurs. For a z800 or z900 processor ICSF software performs the AES key processing. The options include a description for the encrypted data, a request to generate a clear 128-bit AES key with a password. An iteration count is also specified:

```
//ENCRYPT3 EXEC PGM=CSDFILEN
//SYSPRINT DD SYSOUT=*
//SYSUT1   DD PATH='/u/Lab/experiments/test3',PATHOPTS=ORDONLY
//SYSUT2   DD UNIT=3490,DISP=(,KEEP),DSN=LAB.EXP.TEST3,
//         VOL=(RETAIN,,,3)
//SYSIN    DD *
DESC='My Secure Data'
CLRAES128
PASSWORD=1509TY6E
ICOUNT=3200
/*
//
```

ICSF callable services for CSDFILEN

The Encryption Services invokes the following ICSF callable services for CSDFILEN. If you are using RACF or similar security product, ensure that the security administrator authorizes the Encryption Services to use the following services and any cryptographic keys that are input. For information about the ICSF callable services, see *z/OS Cryptographic Services ICSF Application Programmer's Guide*.

- CSFCKM Multiple clear key Import
- CSFENC Encipher
- CSFRNG Generate a random number
- CSFSYE Encipher using clear DES/AES key
- CSFPKE Public key encrypt
- CSFSYG Generate and wrap a symmetric key
- CSFSYX Export a symmetric key
- CSFOWH One-way hash

Encryption Services might use the following ICSF callable service without the need for authorization:

- CSFXBC Convert binary string to character

Chapter 4. Decrypting files through CSDFILDE of the Encryption Services

This chapter presents information about using the CSDFILDE batch program of the Encryption Services to decrypt data.

CSDFILDE is a batch program of the Encryption Services that decrypts encrypted file output from CSDFILEN or Encryption Facility for z/OS Client. You can specify the same options for key protection that were used when CSDFILEN or the Encryption Facility for z/OS Client encrypted the file. You can also specify an option (INFO) to provide information on the original clear key file information and the keys to be used by decryption processing.

JCL DD statements for CSDFILDE

CSDFILDE supports the following DD statements or their dynamic allocation equivalents:

Table 12.

DD statement	Description
SYSPRINT	Specifies the name of the data set to which CSDFILDE writes encryption statistics and diagnostics information. It can be a sequential data set as follows: • Typically a sysout data set • DASD or tape data set • PDS or PDSE member CSDFILDE sets the following values: • RECFM=FBA • LRECL=133 The system selects an optimal value for BLKSIZE unless you choose to code BLKSIZE. BLKSIZE must be a multiple of 133 unless it is a sysout DD statement, but coding BLKSIZE on a sysout DD statement is not beneficial.
SYSIN	Specifies the source from which CSDFILDE reads control statements. It can be a sequential data set as follows: • Typically a spooled system input data set • ONE of the following specifications: – DASD or tape data set – PDS or PDSE member CSDFILDE requires the following specifications: • RECFM=F or FB • LRECL=80
SYSUT1	Specifies the name of the data set that contains the encrypted data to be decrypted. This is the encrypted data from CSDFILEN or the Encryption Facility for z/OS Client. It can be a sequential data set as follows: • DASD or tape data set • PDS or PDSE member If you copy the file that is the output of CSDFILEN to another z/OS file, DO NOT change the LRECL or RECFM DCB parameters.

Table 12. (continued)

DD statement	Description
SYSUT2	Specifies the name of the output data set that contains the decrypted data. It can be a sequential data set as follows: • DASD or tape data set • PDS or PDSE member • z/OS UNIX Systems Services file The output from CSDFILDE can also be a sysout data set (SYSOUT=x). **DISP:** You can code any of the following values for DISP on SYSUT2: • DISP=(NEW,CATLG). The data set did not exist before this job step. After the job step, the system is to catalog and keep the data set. • DISP=(NEW,KEEP). The data set did not exist before this job step. If the new data set is not SMS-managed, the system is to keep the data set but does not catalog it so you must keep track of the volume where the data set is to reside, including disk and tape. If the new data set is SMS-managed, the system treats the data set as if you had coded CATLG. On z/OS DASD data sets are usually SMS-managed. • DISP=(NEW,PASS). The data set did not exist before this job step and a following job step determines the final disposition of the data set. • DISP=(MOD,xxxx) where the xxxx is CATLG, KEEP or PASS as described above. If the data set did not exist before this job step, the system is to allocate the space for the data set. The result is the same as if you had coded DISP=(NEW,xxxx). In this case, you must also code SPACE. If the data set existed before this job step, the system uses it and QSAM adds the new records to the logical end of the existing data set. In this case, SPACE has no effect, but if more space is needed, the system uses the secondary space amount, which is a temporary override for the space. A subsequent program that appends more records if needed uses the originally-coded secondary space amount. **RECFM and LRECL:** You do not need to code RECFM or LRECL for SYSUT2. CSDFILDE assumes that you want to use the original values for RECFM and LRECL for the clear data set. **BLKSIZE and BLKSZLIM:** You do not need to specify BLKSIZE for SYSUT2. Consider the following situations: • If the value for RECFM (record format) for the original data set whose encrypted output is on SYSUT1 is undefined format (RECFM=U), the system calculates an optimal BLKSIZE value for the device represented by the CSDFILDE output data set on SYSUT2. For example, if the original clear data set for SYSUT1 is on tape and the new clear data set for SYSUT2 is on disk, the original block size if specified on SYSUT2 might waste disk space or not work properly. Although you can specify BLKSIZE on SYSUT2 to enforce a specific block size value, the value might be less than optimal. Instead of BLKSIZE, consider coding a value for BLKSZLIM on SYSUT2. When you code BLKSZLIM, the value sets an upper limit on the value for BLKSIZE that the system automatically calculates for the device. Although for tape devices z/OS supports block sizes that are greater than 32760 bytes, many programs (for example, programs that are written before 2004) that need to read the tape data sets do not handle block sizes that are greater than 32760. For any program that cannot handle block sizes greater than 32760, code BLKSZLIM=32760. The value for BLKSZLIM does not have to be a multiple of the LRECL value. • If the RECFM value is undefined format (RECFM=U) on SYSUT1 and you do not code BLKSIZE on SYSUT2, CSDFILDE assumes the block size for the original clear data set. Although this value might not be optimal for the device, CSDFILDE does not attempt to reblock the data.

Control statement keywords for CSDFILDE SYSIN DD

You can specify the following options in the control statement data set (identified by SYSIN DD) to control decryption of the input files. All keywords must start in column 1, and you cannot code a continuation statement. The program treats an asterisk (*) in column 1 as a comment.

All of the following keywords are optional unless the following conditions apply:

- You specified PASSWORD= as input to CSDFILEN. In this case you must supply the correct password on the SYSIN control statement.
- The label of the RSA private key to be used to decrypt the data-encrypting key is different from the label of the RSA public key used by CSDFILEN. In this case you must supply the label of the RSA private key on the JCL statement for CSDFILDE.

Table 13.

Description	JCL keyword
Method to generate and protect the data encrypting key	Specifies the method that is used to generate and protect the data encrypting key. RSA and PASSWORD are mutually exclusive. **RSA=***label* Specifies the 64-byte *label* of an RSA private key that is in the ICSF PKDS. This RSA private key that corresponds to the public key is used to decrypt the data-encrypting key that is present in the header record of the encrypted data from CSDFILEN or Encryption Facility for z/OS Client. If you are decrypting data that is encrypted with a single RSA= control statement and the RSA private key label is the same as the RSA public key label used to protect the data-encrypting key, the RSA keyword is optional because the RSA key label is stored in the header record. For data encrypted with multiple RSA= control statements, you must specify one RSA keyword for decryption on CSDFILDE. CSDFILDE does not allow multiple RSA keywords. See “Specifying multiple RSA keys” on page 21. **PASSWORD=***password* Specifies the 8- to 32-EBCDIC character password to be used to regenerate the clear TDES triple-length key or the clear 128-bit AES key that is used for the data encryption. This password must match that specified as input to CSDFILEN or Encryption Facility for z/OS Client. Passwords are case sensitive. In order to minimize problems because of code page differences at the encrypting and decrypting sites, IBM suggests that you use only the upper and lower-case letters A through Z, numerals 0 – 9 and the underscore character (_) .

Table 13. (continued)

Description	JCL keyword	
Information only	**INFO**	Specifies that file decryption is not to be performed, but that information about the defaults that CSDFILDE establishes is to be recovered and written to the SYSPRINT file. When the information is written, CSDFILDE processing ends. This option is useful when you want to determine the original clear text file DCB information and to ensure that a specified RSA key is present in the current ICSF system.

User reference information for decrypting data

Reference information includes the following topics and samples:

- "Format of the statistics report file for CSDFILDE"
- "Return codes for CSDFILDE" on page 36
- "JCL examples for CSDFILDE" on page 37

Format of the statistics report file for CSDFILDE

The output of the statistics report file depends on whether the encrypted data has been compressed.

COMPRESSION=NO: The following example shows the output of the statistics report file from CSDFILDE when compression has not been specified for the encrypted data:

```
        CSDFILDE Decryption Utility 09/28/2005 (MM/DD/YYYY) 14:41:07 (HH:MM:SS)
 CSDFILDE:            :
INPUT: DESC = DATA TO SEND TO PARTNER
INPUT: LRECL    121 BLKSIZE     484 RECFM FB
INPUT:  PASSWORD=********************************
RECORDS READ:             88  WRITTEN:         22,653
  BYTES READ:         2,877,408  BYTES RECOVERED:         2,741,013
CIPHER TIMES (IN SECONDS): HIGH:   0.001949 DATA:  294840 LOW:   0.001462 DATA:  229320
 TOTAL CIPHER TIME (IN SECONDS):     0.018218  CIPHERS:             10
  TOTAL ELAPSED TIME:   0:00:02.23
```

COMPRESSION=YES: The following example shows the output of the statistics report file from CSDFILDE when compression has been specified for the encrypted data:

```
         CSDFILDE Decryption Utility 09/28/2005 (MM/DD/YYYY) 14:47:50 (HH:MM:SS)
 CSDFILDE:    RSA-PUB : CCA.PVT06.INT.ENC.1024S0F
INPUT: DESC = DATA TO SEND TO PARTNER
INPUT: LRECL    121 BLKSIZE     484 RECFM FB
INPUT:  RSA=CCA.PVT06.INT.ENC.1024S0F
RECORDS READ:             25  WRITTEN:         22,653
  BYTES READ:           789,368  BYTES RECOVERED:         2,741,013
CIPHER TIMES (IN SECONDS): HIGH:   0.001854 DATA:  294376 LOW:   0.001431 DATA:  229320
 TOTAL CIPHER TIME (IN SECONDS):     0.005120  CIPHERS:             3
  TOTAL EXPAND TIME (IN SECONDS):     0.037375 TOTAL EXPANDED BYTES:      2,760,344
  TOTAL ELAPSED TIME:   0:00:09.57
```

If the report indicates 0 for EXPANDED BYTES, an attempt to compress data that is less than 64 K bytes probably occurred with CSDFILEN. See “When should I compress data for encryption?” on page 21

If one or more RSA keywords have been specified for the encrypted data input, and if the INFO keyword is specified as input to CSDFILDE, the output of the statistics report file from CSDFILDE includes all of the RSA key labels (in the previous example, CSDFILDE: RSA-PUB : CCA.PVT06.INT.ENC.1024S0F). If the PASSWORD keyword has been specified for the encrypted data input, and if the INFO keyword is specified as input to CSDFILDE, the output from CSDFILDE in the statistics report file shows blanks in the second line as follows:

```
CSDFILDE Decryption Utility 09/15/2005 (MM/DD/YYYY) 10:51:27 (HH:MM:SS)
CSDFILDE:           :
INPUT: DESC = UR0.B32760 INPUT
INPUT: LRECL     80 BLKSIZE   32720 RECFM FB
INPUT:  *-------------------------------------------------------------------*
INPUT:  *    INFO KEYWORD ONLY                                              *
INPUT:  *-------------------------------------------------------------------*
INPUT:  INFO
```

Understanding the statistics report

In the statistics report, the line that starts CIPHER TIMES (IN SECONDS) shows the longest time (HIGH) that the program takes to decipher a chunk of data and the number of bytes of clear data that are in that chunk. CIPHER TIMES (in seconds) also shows the shortest time (LOW) that the program takes to decipher a chunk of data, and the number of bytes in that chunk. For example if the statistics report contains the following line:

```
CIPHER TIMES (IN SECONDS): HIGH:   0.001854 DATA:  294376 LOW:   0.001431 DATA:  229320
```

the longest amount of time taken for a single decipher is .001854 seconds to decipher a block of 294376 bytes. The shortest amount of time taken for a single decipher is .001431 seconds to decipher a block of 229320 bytes.

In the line that starts TOTAL CIPHER TIME (IN SECONDS):, CIPHERS: indicates the number of times CSDFILDE invokes an ICSF callable service to perform decryption. In the following example, CSDFILEN invoked callable service CSNBDEC 17647 times:

```
TOTAL CIPHER TIME (IN SECONDS):    32.842984  CIPHERS:         17,647
```

The value is not the same as the number of hardware instructions executed because the hardware instruction processes a CPU-determined number of blocks,

each of which are a multiple of the algorithm length (8 bytes for TDES or 16 bytes for AES). The system might require multiple invocations of the hardware instruction to process one ″chunk″ of data that ICSF processes.

CSDFILDE diagnostics

CSDFILDE might write diagnostic information to the statistics report file. Each of the following error lines begins with the characters **ERROR** and ends the processing of CSDFILDE:

```
CONFLICTING OUTPUT LRECL
CONFLICTING OUTPUT RECFM
EOF REACHED ON INPUT FILE BEFORE END OF HEADER
SPECIFY ONE OF RSA/PASSWORD ONLY
PASSWORD MUST BE SPECIFIED
PASSWORD ENTERED NOT VALID
INCORRECT PASSWORD ENTERED
HEADER RECORD NOT VALID
UNEXPECTED ERROR DURING EXPANSION (RETCODE=nn)
SYSUT1 FILE FAILED TO OPEN
SYSUT2 FILE FAILED TO OPEN
UNSUPPORTED VERSION OF ENCRYPTED DATA
OUTPUT MUST SUPPORT QSAM LARGE BLOCK INTERFACE (LBI)
RECORD FORMAT FOR SYSUT1 NOT VALID. RECFM(U) IS REQUIRED.
PASSWORD NOT ALLOWED WITH RSA OPTION
PRODUCT DEREGISTRATION FAILED
```

The following diagnostic information (beginning with the characters **WARNING**) indicates that the output data set (SYSUT2) of CSDFILDE has a different BLKSIZE(nnnnnn) from the input data set (SYSUT1) specified on CSDFILEN. CSDFILDE processing continues:

```
NEW OUTPUT BLKSIZE. REQUESTED: nnnnnn
```

CSDFILDE might also write the following diagnostic information (beginning with the characters **INFO**) to the statistics report file:

```
CSNBOWH possible SAF authorization violation
```

ICSF callable services and diagnostics: If CSDFILDE invokes an ICSF callable service and that service encounters a failure, CSDFILDE writes the following diagnostic information to the statistics report file. In this example, the service CSNDPKD was invoked and returned a return code of 8 and a reason code of X'271C'. For information about ICSF return codes and reason codes, see *z/OS Cryptographic Services ICSF Application Programmer's Guide*.

```
**ERROR**CSNDPKD   08  00271C
```

Corrupted compression dictionary: A system abend code of `0C7` usually means that the compressed data has not been decompressed because either the compression dictionary or the data is corrupted.

Return codes for CSDFILDE

CSDFILDE can issue the following return codes (decimal values in general register 15):

Table 14.

Return code	Meaning
0	Successful operation

Table 14. (continued)

Return code	Meaning
4	Warning
8	User error
20	Physical error occurred on input file
24	Physical error occurred on output file
28	SYSPRINT file failed to open
32	SYSUT1 file failed to open
36	SYSUT2 file failed to open
40	Error reading header from input file
44	Bad data found during expansion of compressed data
48	No expansion dictionary in decrypted input data
50	Product registration/deregistration failed

CSDFILDE can also return the following return codes that the ICSF callable service passes back to the routine. In this case, the return and reason codes from the ICSF service are also recorded in the statistics report file:

Table 15.

Return code	Meaning
4	Warning
8	Application error
12	CSF error
16	Terminating error

JCL examples for CSDFILDE

Example 1: In this example CSDFILDE reads data from a DASD data set named PARTNER.XYZ.ENCDATA. It writes the decrypted data to a DASD data set named PARTNER.XYZ.INVENTORY, which did not previously exist. The system creates and catalogs the data set. All of the defaults are used for the SYSIN options. Because the original data has been encrypted with a CLRTDES key that is protected with an RSA private key and the RSA key label is available in the header, you do not need to specify RSA= if the PKDS of the decrypting system contains the same RSA private key stored with the same label.

```
//DECRYPT EXEC PGM=CSDFILDE
//SYSPRINT DD SYSOUT=*
//SYSUT1 DD DSN=PARTNER.XYZ.ENCDATA,DISP=SHR
//SYSUT2 DD DSN=PARTNER.XYZ.INVENTORY,
//          UNIT=SYSDA,DISP=(NEW,CATLG),
//          SPACE=(1024,(60,10)),AVGREC=K
//SYSIN DD *
/*
//
```

Example 2: In this example CSDFILDE reads encrypted data from an IBM standard labeled tape data set that is named XYZ.FILE.ARCHIV. The data set is also a cataloged data set. CSDFILDE writes the decrypted data to a new tape data set XYZ.DATA, on volume serial ARCHIV. This data set is added to the system catalog. Because the original clear data was encrypted using PASSWORD, you must specify the same password as input to the CSDFILDE program. The program retrieves the

iteration count and salt value used by CSDFILDE from the header record that is contained in the SYSUT1 input file. It uses those values along with the password to recover the clear key for decryption:

```
//DECRYPT2 EXEC PGM=CSDFILDE
//SYSPRINT DD SYSOUT=*
//SYSUT1 DD UNIT=3590,DSN=XYZ.FILE.ARCHIVE,DISP=OLD
//SYSUT2 DD UNIT=3590,DISP=(NEW,CATLG),VOL=SER=ARCHIV,
//          DSN=XYZ.DATA
//SYSIN DD *
PASSWORD=ASK NOT FOR WHOM THE BELL TOLLS
/*
//
```

Example 3: In this example CSDFILDE reads data from a z/OS UNIX Systems Services file named /u/Lab/encdata/partner3. It writes the decrypted data to an existing data set, PARTNER3.INVENTORY, on a 3390 DASD volume named CSDUS1. The original clear data has been encrypted with a key that is protected by an RSA public key. The RSA private key that corresponds to the public key is stored in the PKDS of this system with the label MY.PRIV.2048.KEY. Because this label is different from the label of the RSA public key that has been used for encryption, you must specify the label of the private key for RSA as input to the CSDFILDE program.

```
//DECRYPT3 EXEC PGM=CSDFILDE
//SYSPRINT DD SYSOUT=*
//SYSUT1 DD PATH='/u/Lab/encdata/partner3',PATHOPTS=ORDONLY
//SYSUT2 DD DSN=PARTNER3.INVENTORY,
//          UNIT=3390,VOL=SER=CSDUS1,DISP=OLD
//SYSIN DD *
RSA=MY.PRIV.2048.KEY
/*
//
```

ICSF callable services for CSDFILDE

The Encryption Services invokes the following ICSF callable services for CSDFILDE. If you are using RACF or similar security product, ensure that the security administrator authorizes Encryption Services to use the following services and any cryptographic keys that are specified as input to CSDFILDE. For information about the ICSF callable services, see *z/OS Cryptographic Services ICSF Application Programmer's Guide*.

- CSFCKM Multiple clear key import
- CSFDEC Decipher
- CSFSYD Decipher using clear DES/AES key
- CSFOWH One-way hash
- CSFPKD Public key decrypt
- CSFSYI Import a symmetric key

Encryption Services might use the following ICSF callable service without the need for authorization:

- CSFXBC Convert binary string to character

Chapter 5. Using Encryption Facility for z/OS Client

This chapter presents information about using Encryption Facility for z/OS Client.

Encryption and decryption functions

Encryption Facility for z/OS Client works with the CSDFILEN and CSDFILDE programs to perform the following functions:

- Decrypts data that is encrypted through the CSDFILEN program on z/OS
- Encrypts data that the CSDFILDE program is able to decrypt on z/OS

You can also transfer data that is encrypted by Encryption Facility for z/OS Client to non–z/OS platforms and use Encryption Facility for z/OS Client to decrypt the data.

Installing the Java code

Before you use Encryption Facility for z/OS Client to encrypt or decrypt data on a z/OS system, ensure that you perform the following steps:

1. Download Encryption Facility for z/OS Client zip file from the Java Web site to where you want to run the program. See “Getting started with Encryption Facility for z/OS Client” on page 14.
2. Extract the zip files and place the Java source on the z/OS system.
3. Compile the source code on z/OS.

Java classes: Encryption Facility for z/OS Client includes the following Java classes:

- EncryptionFacility (application with options to encrypt or decrypt a binary file)
- Messages (class that contains the messages for Encryption Facility for z/OS Client)

For complete information about these classes and the use of Encryption Facility for z/OS Client, see the documentation in the JavaDocsPublic directory within the downloadable zip file.

Using RSA keys and certificates

Using RSA keys: When you encrypt data through Encryption Facility for z/OS Client, you can specify multiple RSA public keys through the **-keyStoreCertificateAlias** parameter.

For example, you can use two RSA **-keyStoreCertificateAlias** parameters for `certificate1` and `certficate2` as input for encryption:

```
-keyStoreCertificateAlias="certificate1 alias with imbedded blanks"
-keyStoreCertificateAlias="certificate2 alias with imbedded blanks"
```

You can specify up to 16 RSA key labels for encryption.

When you use Encryption Facility for z/OS Client to decrypt data, use the **-keyStoreCertificateAlias** parameter to specify RSA labels. The decryption process uses the public key portion of the RSA key that is specified by **-keyStoreCertificateAlias** to identify the appropriate encrypted data encryption key in the header. Encryption Facility for z/OS Client then uses the RSA private key to unwrap the data encryption key.

To use RSA keys and certificates with the Java Client you must store your keys in a Java keystore. The most common way to do this is with a utility called `keytool`. For the documentation on the Java keytool utility, see the following Web site:

`http://java.sun.com/j2se/1.4.2/docs/tooldocs/solaris/keytool.html`

Using certificates: Generally speaking, what you need to do with `keytool` depends on how you intend to use the Java Client.

- To send encrypted data to another system, you need the certificate of the other system. For example, the following code shows how to import a certificate (`Acertfile.cer`) from another system into your keystore:

  ```
  keytool -import -alias sysA -file Acertfile.cer
  ```

- To receive encrypted data from another system, the other system needs your certificate. For example, the following code shows how to create a keypair and certificate in your keystore:

  ```
  keytool -genkey -dname "CN=System B, OU=MyUnit, O=MyOrg, L=MyLocal,
  S=MyState, C=MyCountry" -alias sysB
  ```

  The following code shows how to export your certificate (`Bcertfile.cer`) from your keystore to another system:

  ```
  keytool -export -alias sysB -file Bcertfile.cer
  ```

Considerations using Encryption Facility for z/OS Client

Consider the following uses of Encryption Facility for z/OS Client for encrypting or decrypting z/OS format data:

Java policy files: To run the Java application you need to copy the unrestricted Java policy files to your Java Virtual Machine (JVM). This allows you to use large key sizes.

Data compression: Encryption Facility for z/OS Client cannot compress data for encryption. Also, Encryption Facility for z/OS Client cannot process encrypted data that has been compressed by Encryption Facility on z/OS.

Encrypted data from CSDFILEN: Encryption Facility for z/OS Client cannot decrypt data that is encrypted through encrypted keys (that is, if you have specified the ENCTDES keyword on CSDFILEN when you encrypted the data).

Data conversions between systems: Encryption Facility does not handle data conversions that need to take place between z/OS and non-z/OS systems. You must handle all data conversions (for example, adding new line characters or making ASCII, EBCDIC, and other kinds of code conversions).

Editing encrypted files on z/OS: Do not use an editor on z/OS to access a file that has been encrypted with Encryption Facility for z/OS Client, or you might corrupt the file format.

Record format and size: You can use Encryption Facility for z/OS Client to encrypt data in fixed record format. Use **–recordFormat** and **–recordSize** as follows:

- To indicate a fixed record size, specify **-recordFormat** as FIXED and a valid value for **–recordSize**. Allowable record sizes for FIXED record format are from 1 to 32760. Encryption Facility for z/OS Client encrypts the data according to the specified record size and updates the header information to reflect that the data is in fixed record format.

- To indicate an undefined record size, use the default value or specify UNDEFINED for **–recordFormat**. If you use the default value or specify UNDEFINED for **–recordFormat**, you cannot specify a value for **–recordSize**. If you specify a value for **–recordSize**, Encryption Facility for z/OS Client indicates an exception.

Exchanging files between operating systems: If you are exchanging data between the Java file system and z/OS, you must be aware of the file characteristics (record format and length) for z/OS data. For example, if you encrypt a file from z/OS that specifies the following characteristics on the JCL for CSDFILEN

```
RECFM=FB,LRECL=252
```

then send the encrypted file to Encryption Facility for z/OS Client for decryption and use the default or specify UNDEFINED on **–recordFormat**, the Java file system strips out the information from the header. The result is a data stream with records of length 32760. If you encrypt the data on the Java file system and send it back to z/OS, the z/OS system cannot restore the original file characteristics because they are no longer maintained in the header. To use CSDFILDE to decrypt the data on z/OS, you must remember the original file characteristics and specify those characteristics for the JCL on the data control block (DCB) parameter. You can use INFO on CSDFILDE to display the original LRECL and BLKSIZE values that are in the header of the encrypted file.

Chapter 6. Using DFSMSdss Encryption

You can use DFSMSdss Encryption to encrypt and decrypt data through DFSMSdss and DFSMShsm commands. For complete information, see *z/OS DFSMS Storage Administration Reference*.

DFSMShsm documentation

For complete information about using DFSMShsm commands to encrypt and decrypt data, see the following DFSMShsm publications:

- *z/OS DFSMShsm Storage Administration Guide*
- *z/OS DFSMShsm Implementation and Customization Guide*

For DFSMSdss you can use the DUMP command to encrypt an output data set and specify that the encrypted data is to reside on tape or DASD. You can specify the following options on the DUMP command:

Table 16.

Description	DUMP option
Encryption type	**ENCRYPT** Specifies information about which encryption key you want to generate. You can specify one of the following types. **CLRTDES** Specifies that the input file is to be encrypted with a clear TDES triple-length key in the DFSMSdss address space **CLRAES128** Specifies that the input file is to be encrypted with a clear 128-bit AES key ENCTDES in the DFSMSdss address space **ENCTDES** Specifies that the input file is to be encrypted with a secure TDES triple-length key in the DFSMSdss address space

Table 16. (continued)

Description	DUMP option
Method to generate and protect the data encrypting key	Specifies the method to be used to generate and protect the data encrypting key. RSA and PASSWORD are mutually exclusive. One of the following keywords is required: **RSA(*label*)** Specifies the 64-byte *label* of an existing RSA public key that is present in the ICSF cryptographic key data set (PKDS). **KEYPASSWORD(*password*)** Specifies a password between 8 and 32 characters that is used to generate a data key to encrypt the user data. If you specify KEYPASSWORD on the DUMP command, you must also specify the same KEYPASSWORD on the RESTORE command. IBM suggests that you use only the upper and lower-case letters A through Z, numerals 0 – 9 and the underscore character (_) .
Compression option	**HWCOMPRESS** Specifies whether you want compression of the clear input before encryption of the data occurs. If you want compression, specify the keyword HWCOMPRESS. Omit the keyword if you do not want compression.

To decrypt the data from the DUMP command, you can use the RESTORE command with the following options:

Table 17.

Description	RESTORE option
Method used to generate and protect the data encrypting key	Specifies the method to be used to generate and protect the data encrypting key. RSA and PASSWORD are mutually exclusive: **RSA(*label*)** Specifies the 64-byte *label* of an existing RSA private key that is present in the ICSF PKDS. The RSA option on the RESTORE command is optional. Use RSA if you want to specify a different label for an RSA key. If you do not specify the RSA keyword on the RESTORE command, DFSMSdss uses the original label specified on the DUMP command. **KEYPASSWORD(*password*)** Specifies a password between 8 and 32 characters that is used to generate a data key to encrypt the user data. If KEYPASSWORD has been specified on the DUMP command, you must also specify the same KEYPASSWORD on the RESTORE command. IBM suggests that you use only the upper and lower-case letters A through Z, numerals 0 – 9 and the underscore character (_) .

JCL examples for DFSMSdss Encryption

Example 1: This example shows the JCL for encrypting data for a full volume dump to tape:

```
//SYSIN    DD *
  DUMP -
       FULL -
           INDD ( -
               DC9SS01  -
               ) -
           ENCRYPT(CLRAES128) -
           HWCOMPRESS -
           RSA(CCA.CRT08.INT.ENC.1024S0F) -
           OUTDDNAME (DDTAPE1)
/*
```

Example2: This example shows the JCL for encrypting a logical dump data set and decrypting the data through the RESTORE command:

```
//SYSIN    DD *
  DUMP -
       DS(INCLUDE(SOURCE.**)) -
           LOGINDD ( -
               DC9SS01  -
               DC9SS02  -
               ) -
           ENCRYPT(CLRTDES) -
           KEYPASSWORD(MYPASSWORD) -
           HWCOMPRESS -
           ALLDATA(*) -
           ALLEXCP -
           OUTDDNAME (DDTAPE1)
/*
//SYSIN    DD *
  RESTORE -
          DS(INCLUDE(SOURCE.**)) -
    OUTDYNAM ( -
    (T9SS01) -
    (T9SS02) -
    (T9SS03) -
     ) -
    KEYPASSWORD(MYPASSWORD) -
    RENAMEU(TARGET) -
    REPLACEU -
    STORCLAS(SC9TG016) -
    INDDNAME (DDTAPE1)
/*
```

Chapter 7. Using RACF with Encryption Facility

You can use RACF to help you store RSA public and private keys for encryption in the ICSF public key data set (PKDS). You can also specify the PKDS labels to use when you store public or private keys and can list PKDS labels of existing security certificates as well as establish a PKI to manage the digital certificate authority for users.

This chapter includes the following topics:

RACF APAR

To use the RACF enhancements for this release of Encryption Facility, ensure that you have the PTF for APAR OA13030 installed on z/OS 1.4 or later.

Using RACF to store keys, manage PKDS labels, and send digital certificates

Encryption Facility encrypts data for archival and recovery purposes. Long term storage of archived encrypted data helps with disaster recovery. Moreover, Encryption Facility allows you to safely transport encrypted data to interested parties on other sites where it can be decrypted or stored for future use. RACF provides public/private key support and the management of PKDS labels associated with the keys for both archiving and recovery of the encrypted data. Through RACF, you can also set up a limited scope certificate authority that allows you to exchange key information for the encrypted data.

For data archival and recovery, the encryption and decryption process can make use of the RSA public/private key pair. The public part is meant for encrypting data that can be archived, and the private part for decrypting or recovering the data.

For data transit, the sending side needs to use the recipient's public key to encrypt the data, while the receiving side uses the corresponding private key of the public/private pair to decrypt the data. The sender expects to receive the public key of the recipient in the form of an x.509 certificate, and the sender needs to store the public key, without the equivalent private key, in the ICSF PKDS.

If you are using RSA private/public keys to encrypt or decrypt data, you can make use of RACF to allow you to exchange keys with the senders or recipients of the data. This exchange involves using digital certificates to identify users and their keys. You can also use ICSF utility panels to create or delete PKDS records and export or import RSA keys to x.509 certificates. For a scenario, see "Using ICSF utilities panels for PKDS key management" on page 65. For complete information, see *z/OS Cryptographic Services ICSF Administrator's Guide*.

RACF and certificates: RACF lets you create certificates with a private/public RSA key pair for the decryption of data sent to you by another party. To accomplish this,

the sender must encrypt the data using your public key. Before encryption, this public key needs to be sent to the other party enclosed in the certificate created by RACF.

This certificate identifies you as the owner of the key pair. The other party can receive the certificate and use RACF to create a PKDS label for the public key in the other party's ICSF PKDS. In addition, if you want to encrypt data to return to the other party, a second key exchange must occur, where the other party creates the key pair and sends you the public key certificate from that system.

Figure 3 shows how this exchange of certificates works with RSA keys that are stored in the PKDS of the sending and receiving systems. On z/OS you can use RACF to generate a digital certificate and key pair. You (the z/OS customer) send the digital certificate to a receiver (the business partner) on a remote system who can also use RACF or another program or service to create a digital certificate and key pair and send the certificate to you.

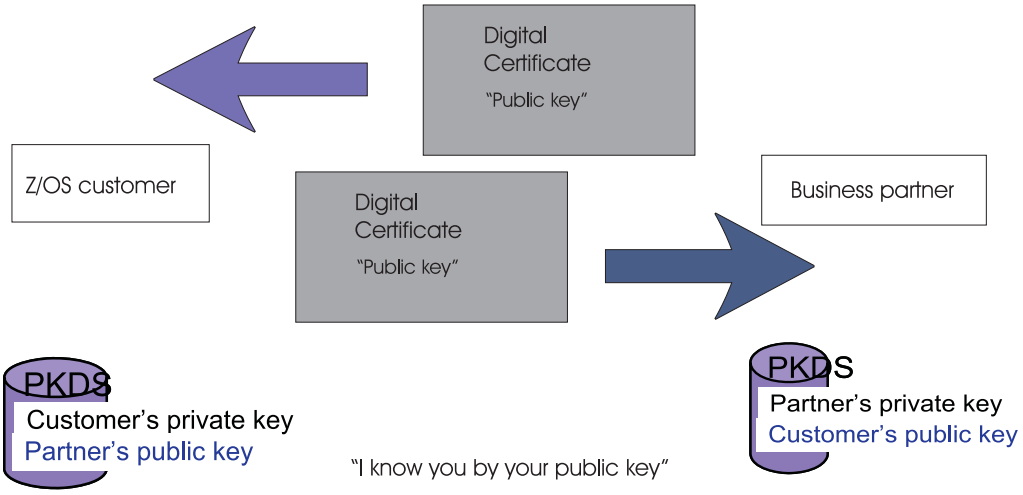

Figure 3. Establishing a trusted exchange through digital certificates

Using the RACDCERT command

RACDCERT is used to install and maintain digital certificates, key rings, and digital certificate mappings in RACF. RACDCERT should be used for all maintenance of the DIGTCERT, DIGTRING, and DIGTNMAP class profiles. For complete information, see *z/OS Security Server RACF Command Language Reference*.

For Encryption Facility, RACF provides the following support for the RACDCERT command:

- RACDCERT ADD includes a new keyword PCICC in addition to the ICSF keyword to allow user-defined PKDS labels.
- Changes exist to the description for RACDCERT GENCERT PCICC, ICSF, and DSA keywords
- Changes exist to the description of the RACDCERT REKEY PCICC and ICSF keywords.

```
RACDCERT ADD(data-set-name)
  [TRUST | NOTRUST | HIGHTRUST]
  [WITHLABEL('label-name')]
  [PASSWORD('pkcs12-password')]
  [ICSF[(pkds-label | *)] | PCICC[(pkds-label | *)] ]
```

[ICSF[(*pkds-label* **| *)] | PCICC[(***pkds-label* **| *)]**

Specify that RACF should store the public or private key associated with this certificate in the ICSF PKDS. This includes when the key is introduced to RACF by issuing the ADD keyword and when an existing certificate profile is replaced by issuing the ADD keyword. You can specify the PKDS label to be assigned to the key (optional), or specify * if the PKDS label should be taken from the WITHLABEL keyword. In either case, the PKDS label must be unique and meet the ICSF syntax requirements (see notes below).

These keywords only apply to public and private RSA keys created outside of your ICSF environment.

Either keyword is ignored if the key uses the DSA algorithm.

If the certificate being added has an associated private key, the following action takes place:

- If you specify ICSF, RACF stores the private key as an ICSF RSA Modulus-Exponent key token.
- If you specify PCICC, RACF stores the private key as an ICSF RSA Chinese Remainder Theorem (CRT) key token using the PCI cryptographic coprocessor.

The latter is desirable for performance reasons and required for RSA private keys between 1025 and 2048 bits in length. Certificates with private keys greater than 2048 bits in length cannot be processed by RACF. For either keyword, if ICSF is not operational or is not configured for PKA operations, processing stops and RACF displays an error message. For PCICC, a PCI class cryptographic coprocessor must also be present and operational.

If you do not specify the PKDS label (or *), RACF generates a label in the format `IRR.DIGTCERT.userid.cvtsname.ebcdic-stck-value`, where userid is the owning user ID, `cvtsname` is the system name (taken from the CVT), and `ebcdic-stck-value` is an EBCDIC version of the current store clock value.

If the certificate being added does not have an associated private key (that is, it only has a public key), the following action takes place:

- If you specify either keyword and you also specify the PKDS label (or *), RACF stores the public key as an ICSF RSA Modulus-Exponent (ME) key token.
- If you specify either keyword and you do not specify the PKDS label (or *), RACF ignores the keyword (that is, a PKDS entry is NOT created).

For either keyword, if ICSF is not operational or is not configured for PKA operations, processing stops and RACF displays an error message.

Notes:

1. If the private key is stored as an ICSF or PCICC key, any system using the key in the future is required to have ICSF operational and configured for

PKA operations with this PKDS. This includes SSL applications. Use of a PCICC key also requires an operational PCI class cryptographic coprocessor.

2. The default action for a new key is to store it as a software key. The default action for an existing key is to leave it unchanged.
3. If the public or private key already exists in the PKDS, you cannot change its PKDS label by using the ICSF or PCICC keywords to respecify the label.
4. If the private key already exists in the PKDS as an ICSF key, specifying PCICC does not convert the key to a PCICC key.
5. If the private key already exists in the PKDS as a PCICC key, specifying ICSF does not convert the key to an ICSF key.
6. If the public key already exists in the PKDS, and its matching private key certificate is added, you must specify the ICSF or PCICC keyword. This specification upgrades the PKDS entry to contain the ICSF or PCICC private key.
7. If the PKDS label is specified with an asterisk (*), you must also specify the WITHLABEL keyword.
8. If the PKDS label is specified or taken from the WITHLABEL keyword, it must conform to ICSF label syntax rules as follows: Allowed characters are alphanumeric, national (@,#,$) or period (.). The first character must be alphabetic or national. The label has a maximum length of 64 characters and is case insensitive (folded to upper case).

```
RACDCERT GENCERT[(request-data-set-name)]
 [SUBJECTSDN([C('Country')]
            [SP('State or Province')]
            [L('Locality')]
            [O('Organization Name')]
            [OU('Organizational Unit Name 1'
                [,'Organizational Unit Name 2']
                [,'Organizational Unit Name n'])]
            [T('Title')]
            [CN('Common Name')]]
 [SIZE(Key Size)]
 [NOTBEFORE([DATE(yyyy-mm-dd)] [TIME(hh:mm:ss)])]
 [NOTAFTER([DATE(yyyy-mm-dd)] [TIME(hh:mm:ss)])]
 [WITHLABEL('Label Name')]
 [SIGNWITH([SITE | CERTAUTH]
          LABEL('Label Name'))]
 [KEYUSAGE(HANDSHAKE
          DATAENCRYPT
          DOCSIGN
          CERTSIGN)]
 [ALTNAME([IP(Numeric-IP-Address)]
         [EMAIL('Email Address')]
         [DOMAIN(Internet-Domain-Name)]
         [URI(Universal-Resource-Identifier)]]
 [ICSF[(pkds-label | *)] | PCICC[(pkds-label | *)] | DSA ]
```

[ICSF[(*pkds-label* | *)] | PCICC[(*pkds-label* | *)] | DSA]
Specifies how RACF should generate the key pair and how the key should be stored for future use.

If GENCERT is issued without a `request-data-set-name`, the following action takes place:

- If you specify PCICC, RACF generates the key pair using a PCI class cryptographic coprocessor.
- If you specify ICSF, RACF generates the key pair using software.

In either case, the resulting private key is generated with the RSA algorithm and stored in the ICSF PKDS. For ICSF, RACF stores the private key as an ICSF RSA Modulus-Exponent key token. For PCICC, RACF stores the private key as an ICSF RSA Chinese Remainder Theorem (CRT) key token. The latter is desirable for performance reasons and required for RSA private keys between 1025 and 2048 bits in length.

You can optionally specify the PKDS label to be assigned to the key or specify an asterisk (*) if the PKDS label should be taken from the WITHLABEL keyword. In either case, the PKDS label must be unique and meet the ICSF syntax requirements (see notes below).

If either the PKDS label or asterisk (*) is not specified, RACF generates a label in the format `IRR.DIGTCERT.userid.cvtsname.ebcdic-stck-value`, where userid is the owning user ID, `cvtsname` is the system name (from the CVT), and `ebcdic-stck-value` is an EBCDIC version of the current store clock value.

Note: If DSA is specified, the key pair is generated using software with DSA algorithm, and RACF stores the private key in the RACF database as a non-ICSF DSA key. DSA key generation can be very slow, especially for keys greater than 1024 bits.

If you do not specify either of these keywords, RACF generates the key pair using software with the RSA algorithm and stores the private key in the RACF database as a non-ICSF key. If you specify either keyword PCICC or ICSF and ICSF is not operational or is not configured for PKA operations, processing stops and RACF displays an error message. If you specify the PCICC keyword, you must ensure that a PCI class cryptographic coprocessor is present and operational. Otherwise, processing stops and RACF displays an error message.

If you issue GENCERT with a `request-data-set-name`, the following action takes place:

- RACF does not generate a key-pair. The public key from the request is used in the generated certificate. After the certificate is generated, processing for the ICSF and PCICC keywords follows the same rules as those for RACDCERT ADD.
- If you specify the DSA keyword with a `request-data-set-name`, RACF ignores the DSA keyword.

Notes:

1. The DSA keyword is not available on z/OS V1.4, 1.5, or 1.6.
2. If the private key is stored as an ICSF or PCICC key, any system using the key in the future is required to have ICSF operational and configured for PKA operations with this PKDS. This includes SSL applications. Use of a PCICC key also requires an operational PCI class cryptographic coprocessor.
3. The default action for a new key is to store it as a software key.
4. If the public or private key already exists in the PKDS, you cannot change its PKDS label by using the ICSF or PCICC keywords to respecify the label.
5. If the private key already exists in the PKDS as an ICSF key, specifying PCICC does not convert the key to a PCICC key.

6. If the private key already exists in the PKDS as a PCICC key, specifying ICSF does not convert the key to an ICSF key.
7. If the PKDS label is specified with an asterisk (*), you must also specify the WITHLABEL keyword.
8. If the PKDS label is specified or taken from the WITHLABEL keyword, it must conform to ICSF label syntax rules as follows: Allowed characters are alphanumeric, national (@,#,$) or period (.). The first character must be alphabetic or national. The label has a maximum length of 64 characters and is case insensitive (folded to upper case).

```
RACDCERT REKEY
 [SIZE(Key Size)]
 [NOTBEFORE([DATE(yyyy-mm-dd)] [TIME(hh:mm:ss)])]
 [NOTAFTER([DATE(yyyy-mm-dd)] [TIME(hh:mm:ss)])]
 [WITHLABEL('Label Name')]
 [ICSF[(pkds-label | *)] | PCICC[(pkds-label | *)] ]
```

[ICSF[(*pkds-label* **| *)] | PCICC[(***pkds-label* **| *)]]**

Specifies how RACF should generate the key pair and how the private key should be stored for future use if the key algorithm of the original certificate is RSA. If you specify PCICC, the key pair is generated using a PCI class cryptographic coprocessor. If you specify ICSF, the key pair is generated using software. In either case the resulting private key is generated with the RSA algorithm, and RACF stores it in the ICSF PKDS.

For ICSF, RACF stores the private key as an ICSF RSA Modulus-Exponent key token. For PCICC, RACF stores the private key as an ICSF RSA Chinese Remainder Theorem (CRT) key token. The latter is desirable for performance reasons and required for RSA private keys between 1025 and 2048 bits in length.

You can optionally specify the PKDS label to be assigned to the key, or specify an asterisk (*) if the PKDS label should be taken from the WITHLABEL keyword. In either case, the PKDS label must be unique and meet the ICSF syntax requirements (see notes below).

If the key algorithm of the original certificate is DSA, the ICSF and PCICC keywords are ignored. RACF generates the key pair using software with the DSA algorithm and stores the private key in the RACF database as a non-ICSF DSA key.

If you do not specify either keyword, and if the key algorithm of the original certificate is RSA, RACF generate the key pair using software with the RSA algorithm and stores the private key in the RACF database as a non-ICSF key. If you do not specify either keyword and ICSF is not operational or is not configured for PKA operations, processing stops, and RACF displays an error message. If you specify the PCICC keyword, a PCI class cryptographic coprocessor must also be present and operational. Otherwise, processing stops, and RACF displays an error message.

Notes:

1. The REKEY keyword is not available on z/OS V1.4 or 1.5.
2. If the private key is stored as an ICSF or PCICC key, any system using the key in the future is required to have ICSF operational and configured for

PKA operations with this PKDS. This includes SSL applications. Use of a PCICC key also requires an operational PCI class cryptographic coprocessor.

3. The default action for a new key is to store it as a software key.
4. If the PKDS label is specified as *, you must also specify the WITHLABEL keyword.
5. If the PKDS label is specified or taken from the WITHLABEL keyword, it must conform to ICSF label syntax rules as follows: Allowed characters are alphanumeric, national (@,#,$) or period (.). The first character must be alphabetic or national. The label has a maximum length of 64 characters and is case insensitive (folded to upper case).

```
LIST [ (LABEL('label-name')) ] | [ (SERIALNUMBER(serial-number) [ ISSUERSDN('issuer's-dist-name')]) ]
```

LIST [(LABEL('*label-name***'))] | [(SERIALNUMBER(***serial-number***) [ISSUERSDN('***issuer's-dist-name***')])]**

Displays the digital certificate information, including certificate authority and site certificate information. For each digital certificate defined, the following information is displayed:

- Label
- Certificate ID
- Status (trusted, not trusted, or highly trusted)
- Validity dates
- Serial number
- Issuer's distinguished name
- Up to 256 bytes of the subject's name, as found in the certificate itself
- Extensions, if present (specifically, keyUsage and subjectAltName)
- Type of private key (ICSF, non-ICSF or PCICC), or NONE if there is no private key
- Private key size
- PKDS label if the public or private key is stored in the ICSF PKDS
- Ring associations, if present (the ring name to which this certificate is connected and the ring owner)

The following sample is output from the RACDCERT LIST command that shows the PKDS Label IRR.DIGTCERT.GEORGEM.SY1.BD7103108611F42F:

```
Digital certificate information for user GEORGEM:
   Label: New Cert Type - Ser # 00
   Certificate ID: 2QfHxdbZx8XU1YWmQMOFmaNA46iXhUBgQOKFmUB7QPDw
   Status: TRUST
   Start Date: 1996/04/18 03:01:13
   End Date:   1998/02/13 03:01:13
   Serial Number:
       >00<
   Issuer's Name:
       >OU=Internet Demo CertAuth.O=TheCert Software Inc.<
   Subject's Name:
       >OU=Internet Demo CertAuth.O=TheCert Software Inc.<
   Private Key Type: ICSF
   Private Key Size: 1024
   PKDS Label: IRR.DIGTCERT.GEORGEM.SY1.BD7103108611F42F
   Ring Associations:
   Ring Owner: GEORGEM
   Ring:
      >GEORGEMsNewRing01<
   Ring Owner: GEORGEM
   Ring:
      >GEORGEMsRing<
```

See “Using the RACDCERT command for key and certificate management of encrypted data” on page 64.

Considerations using RACDCERT

RACF database and PKDS in a sysplex: If you are sharing a RACF database in a sysplex environment, you must ensure that you are also sharing the PKDS where Encryption Facility resides. For example, if you use RACDCERT GENCERT to generate an RSA key pair and you try to load the public key into the PKDS of another partition that does not share the RACF database, RACF fails the command.

Archiving RSA key data: If you use RACF to create private key entries in the PKDS for use with Encryption Facility, do not issue the following RACF commands that might delete the PKDS entries:

- RACDCERT DELETE deletes a certificate from RACF and its corresponding entry in the PKDS.
- DELUSER deletes any certificates owned by the user and any corresponding entries in the PKDS.
- RACDCERT ROLLOVER command retires a certificate’s private key and deletes its entry in the PKDS

As a precaution, backup the certificate and its private key **before** you place it in the ICSF PKDS as follows:

1. Generate the certificate and key pair.
2. Export the pair to a data set in PKCS12 format (password protected).
3. Migrate the key to ICSF.

The following example shows the RACF commands you can use:

```
RACDCERT GENCERT... /* Do not specify the ICSF, PCICC, or DSA keywords */
RACDCERT EXPORT(LABEL('cert-label')) DSN(backup-data-set-name) FORMAT(PKCS12DER) PASSWORD('secret-password')
RACDCERT ADD(backup-data-set-name) PASSWORD('secret-password') ICSF(pkds-label) | PCICC(pkds-label)
```

ICSF and RACDCERT: RACDCERT processing makes use of ICSF services. If your installation has established access control over ICSF services, the issuers of RACDCERT need to be granted READ authority to ICSF services as follows:

Table 18. RACDCERT command authority and ICSF services

RACDCERT command	Keywords	ICSF Service	Comments
GENCERT or REKEY		• CSFRNG	Only required if ICSF is active
GENCERT or REKEY	PCICC	• CSFPKG • CSFPKX	
GENCERT or REKEY	ICSF or PCICC	• CSFPKRC • CSFPKRR • CSFPKRW	
GENCERT	SIGNWITH	• CSFDSG	Only required if the signing certificate has an ICSF or PCICC key
DELETE		• CSFPKRD	Only required if the certificate being deleted has an ICSF or PCICC key

RACF messages and codes for Encryption Facility

The following messages might appear when you use the RACF support for Encryption Facility:

IRRD159I The key size requires the use of a PCI Cryptographic Coprocessor. The PCICC keyword must be specified. The request is not processed.

Explanation: You are attempting to generate or add a certificate and its private key where the private key is to be generated or saved as a software key. The private key's size is larger than what is allowed as a software key and cannot be processed. However, RACF has detected the presence of a PCI cryptographic coprocessor (PCICC). The private key's size may be acceptable if processed as a PCICC key.

System action: The command is not processed.

User response: If you are generating a certificate and a software key is required, reissue the command with a smaller key size. Otherwise, if a PCICC key is acceptable, reissue the command specifying PCICC. For more information, see *z/OS Security Server RACF Command Language Reference*.

RRD160I WITHLABEL value cannot be used as a PKDS label .

Explanation: You are attempting to generate, add, or rekey a certificate and store its key in the ICSF PKDS. The WITHLABEL keyword was specified along with either ICSF(*) or PCICC(*), indicating that the WITHLABEL value should also be used for the PKDS label. The WITHLABEL does not meet the syntax requirements for a PKDS label. The allowed characters are alphanumeric, national (@,#,$) or period (.). Additionally, the first character must be alphabetic or national.

System action: The command is not processed.

User response: Reissue the command with a WITHLABEL value that meets ICSF requirements or keep the WITHLABEL value you have and specify a different value for the PKDS label in place of the asterisk. For more information, see *z/OS Security Server RACF Command Language Reference*.

IRRD161I The certificate cannot be {added | generated}. The PKDS label 'pkds-label-value' is already in use.

Explanation: You are attempting to generate, add, or rekey a certificate and store its key in the ICSF PKDS. The PKDS label value specified has already been assigned to another PKDS entry. No two entries in the PKDS can have the same label.

System action: The command is not processed.

User response: Reissue the command with a different PKDS label value. For more information, see *z/OS Security Server RACF Command Language Reference*.

IRRD162I The certificate cannot be {added | generated}. The certificate's key is already stored under PKDS label 'pkds-label-value'.

Explanation: You are attempting to renew or readd a certificate and store its key in the ICSF PKDS. The certificate's key has already been saved in the PKDS with the label displayed in the error message. The PKDS label may not be respecified.

System action: The command is not processed.

User response: Reissue the command without specifying a PKDS label value. For more information, see *z/OS Security Server RACF Command Language Reference*.

Chapter 8. User scenarios

The following scenarios show some ways that you can use Encryption Facility to encrypt and decrypt files and RACF to perform certificate exchange.

- "Encrypting data using z/OS and decrypting using Encryption Facility for z/OS Client"
- "Using the RACDCERT command for key and certificate management of encrypted data" on page 64
- "Using ICSF utilities panels for PKDS key management" on page 65

Also, see "JCL Examples for CSDFILEN" on page 28, "JCL examples for CSDFILDE" on page 37, and "JCL examples for DFSMSdss Encryption" on page 45.

Encrypting data using z/OS and decrypting using Encryption Facility for z/OS Client

The following scenarios describe encrypting files on z/OS and decrypting the files on Encryption Facility for z/OS Client.

Scenario 1

In this scenario the JCL for CSDFILEN on z/OS specifies that the data is to be encrypted using a password and a clear 128–bit AES key (CLRAES128). An iteration count (ICOUNT) of 9 is also specified. On z/OS, the device type is a 3390, the data set to be encrypted is a member of a partitioned data set (PO) with a record format of FB, record length of 80, and a block size of 23440. The user saves the encrypted file in the hierarchical file system (HFS) that an xSeries® system can access.

```
//TESRENC   JOB ((ENCRYPT,UC),'ZEF.TEST',
//          NOTIFY=,MSGLEVEL=(1,1),MSGCLASS=H,
//          REGION=4M
//*
//************************************************************
//*
//*  FOLLOWING JOB TAKES DATA FROM ONE DSN AND ENCRYPTS IT
//*  AND PLACES THE RESULT IN ANOTHER DATASET
//*
//*
//************************************************************
//*-----------------------------------------------------------
//*        ENCRYPT
//*      (ONLY MANDATORY ENTRY IS EITHER RSA OR PASSWORD)
//*
//*-----------------------------------------------------------
//ENC      EXEC PGM=CSDFILEN
//SYSPRINT DD SYSOUT=*
//*----------
//STEPLIB  DD DSN=ICSFTST.HCF7730.LINKLIB,DISP=SHR
//SYSUDUMP DD SYSOUT=*
//SYSOUT   DD SYSOUT=*
//SYSIN    DD *
DESC='ENCRYPTION, PW length of 9 Mixedcase, fb data to java x-series'
PASSWORD=pwOFNINE9
ICOUNT=9
CLRAES128
  /*
//SYSUT1   DD DSN=TSTFLR6.FXT.CLIST(AMIOUT),DISP=SHR
//SYSUT2   DD DSN=PAYROLL.G118.FB80.TOJEFF.SEP29T1,
//            UNIT=SYSDA,DISP=(NEW,CATLG),
//            SPACE=(1024,(60,10))
/*
```

The user on the z/OS system copies the encrypted file to the HFS with the file name PAYROLL.ZSERIES.ENCRYPTD (not shown).

On the Java statements for Encryption Facility for z/OS Client, the user on the xSeries system specifies the same password used to encrypt the data to perform decryption as follows. In this scenario Encryption Facility for z/OS Client automatically detects the key used for the encryption and the iteration count:

Table 19.

```
java -Djava.encryption.facility.debuglevel=0 \
com.ibm.encryptionfacility.EncryptionFacility \
-mode decrypt \
-password "pwOFNINE9" \
-inputFile PAYROLL.ZSERIES.ENCRYPTD \
-outputFile PAYROLL.XSERIES.DECRYPTD
```

Scenario 2

In this scenario, two files are encrypted through CSDFILEN on a z/OS system. The files are encrypted using a CLRTDES data-encryption key. One file protects the data-encryption key with an RSA public key extracted from the ICSF PKDS. The other file uses a password to generate the data-encryption key used. The encrypted files are sent to a system where Encryption Facility for z/OS Client decrypts the files.

Using RSA to encrypt on z/OS: CSDFILEN encrypts a file using a CLRTDES key data-encrypting key. The data-encrypting key is then encrypted with an existing RSA public key that is retrieved from ICSF PKDS with the label specified by `RSA=`.

```
//*-------------------------------------------------------------
//ENC      EXEC PGM=CSDFILEN
//*----------
//SYSUDUMP DD SYSOUT=*
//SYSOUT   DD SYSOUT=*
//*----------
//SYSPRINT DD DSN=USERID.ZEF.LISTING,DISP=(MOD,PASS)
//*----------
//SYSUT1   DD DSN=USERID.ZEF.CLR,DISP=SHR
//SYSUT2   DD DSN=USERID.ZEF.RSA,ENC,DISP=SHR
//SYSIN    DD *
*-------------------------------------------------------------------*
*    RSA key encryption on z                                        *
*-------------------------------------------------------------------*
DESC='TDES ENCRYPTED FILES  USING 1023 BIT RSA KEY '
RSA=IRR.DIGTCERT.ENICHEN.SYS1.BDA7284CD8F33AB2
CLRTDES
/*
```

Using PASSWORD to encrypt on z/OS: CSDFILEN encrypts a file using a CLRTDES data-encrypting key. The data-encrypting key is generated using the specified password.

```
//*-------------------------------------------------------------
//ENC      EXEC PGM=CSDFILEN
//*----------
//SYSUDUMP DD SYSOUT=*
//SYSOUT   DD SYSOUT=*
//*----------
//SYSPRINT DD DSN=USERID.ZEF2.LISTING,DISP=(MOD,PASS)
//*----------
//SYSUT1   DD DSN=USERID.ZEF2.CLR,DISP=SHR
//SYSUT2   DD DSN=USERID.ZEF2.PW.ENC,DISP=SHR
//SYSIN    DD *
*-------------------------------------------------------------------*
*    PASSWORD encryption on a z                                     *
*-------------------------------------------------------------------*
DESC='TDES ENCRYPTED FILES  USING Password '
PASSWORD=ABCD1234
CLRTDES
/*
```

Using RSA keys and certificates on Encryption Facility for z/OS Client: To use RSA keys and certificates with Encryption Facility for z/OS Client, you must store your keys in a Java keystore. The most common way to do this is with a utility called `keytool`. To transfer the RSA public key between systems, you must use X.509 certificates. For information about using Encryption Facility for z/OS Client, see the README file documentation from the following Web site: http://www.ibm.com/servers/eserver/zseries/zos/downloads/#asis.

Decrypting the file protected with RSA: The recipient of the encrypted file copies the file into `zef.rsa.enc`. Encryption Facility for z/OS Client is used to decrypt the RSA protected file. The data-encrypting key is recovered using an RSA private key from the Java keystore. The KEYPW on the export command specifies a password used to recover the RSA key:

```
 export MODE='decrypt'
 export KEYSTORETYPE='jks'
 export KEYSTORENAME='/home/g1a5445/edar/test/keystores/keystore_jks'
 export ALIAS_1='peggyrac'
 export KEYPW='xyz12abc'

java  com.ibm.encryptionfacility.EncryptionFacility \
-mode $MODE \
-password $KEYPW \
-keyStoreType $KEYSTORETYPE \
-keyStoreName $KEYSTORENAME \
-keyStoreCertificateALIAS $ALIAS_1 \
-inputFile  zef.rsa.enc \
-outputFile zef.rsa.dec
```

Decrypting the file protected with PASSWORD: The recipient of the encrypted file copies the file into `zef.pw.enc`. Encryption Facility for z/OS Client is used to decrypt the password protected file. The data-encrypting key is regenerated using the password provided in the Java option: `-password $PASSWORD`. The password is the same password that was used to encrypt the original file.

```
export MODE='decrypt'
export PASSWORD='ABCD1234'

java  com.ibm.encryptionfacility.EncryptionFacility \
-mode $MODE \
-password $PASSWORD \
-inputFile  zef.pw.enc \
-outputFile zef.pw.dec
```

Scenario 3

In this scenario, a z/OS system that has access to a UNIX Systems Services (USS) file system creates an encrypted data set on tape and copies the encrypted data set to a USS file. USS then invokes a batch job for Encryption Facility for z/OS Client to decrypt the file to another USS file and copies the decrypted USS file to a z/OS data set.

Encrypting data using z/OS and placing the encrypted data in a tape data set: The following JCL for CSDFILEN encrypts the z/OS data set `HLQ.INPUT.DATASET` and places the output into a tape data set. CSDFILEN uses a password (`TEST1PASSWORD`) to encrypt the data and places the encrypted data in the data set `HLQ.ENCRYPTED.TAPE.DATASET`:

```
//ESE2TAPE JOB 'JOB INFORMATION','TEST',MSGLEVEL=(1,1),
// MSGCLASS=H,CLASS=A,NOTIFY=system_id,REGION=4M
//*****************************************************************
//* This is sample JCL which can be used to encrypt data and place *
//* it into a tape dataset using the Encryption Services provided *
//* by the IBM Encryption Facility for z/OS *
//* *
//* This job uses the CSDFILEN batch program of the Encryption *
//* Services to encrypt the data to tape. The SYSUT1 DD statement *
//* specifies the name of the dataset that contains the data to *
//* be encrypted. The SYSUT2 DD statement contains the name of the *
//* tape dataset that will contain the encrypted data. *
//* *
//* Refer to the IBM Encryption Facility for z/OS: User's Guide for *
//* additional information. *
//*****************************************************************
//ENC EXEC PGM=CSDFILEN
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSOUT DD SYSOUT=*
//SYSIN DD *
PASSWORD=TEST1PASSWORD
//SYSUT1 DD DSN=HLQ.INPUT.DATASET,DISP=SHR
//SYSUT2 DD UNIT=3590-1,DSN=HLQ.ENCRYPTED.TAPE.DATASET,
// DISP=OLD,VOL=SER=ABCDEF
/*
```

Copying the data to USS and using batch services to invoke Encryption Facility for z/OS Client: The JCL performs the following steps:

1. In `STEP1` the IEBGENER copy program on z/OS copies the encrypted data set `HLQ.ENCRYPTED.TAPE.DATASET` from the tape and places it into a USS file called `/filesys/input.encrypt.file` on the USS file system.
2. In `STEP2` a BPXBATCH program on USS runs the shell script `javadecrypt_pw.sh` to invoke Encryption Facility for z/OS Client that decrypts the data in the USS file `/filesys/input.encrypt.file`. The JCL for BPXBATCH specifies that the decrypted file is to be placed into another USS file `/filesys/output.decrypt.file`.

   For the sample script that Encryption Facility for z/OS Client uses to decrypt the data, see “Running the Encryption Facility for z/OS Client sample script” on page 63.
3. In `STEP3` IEBGENER copies the decrypted USS file `/filesys/output.decrypt.file` from `STEP2` to a z/OS data set `HLQ.DECRYPT.OUTPUT`.

```
//JCD2DSN JOB 'JOB INFORMATION',MSGLEVEL=(1,1),
// MSGCLASS=H,CLASS=A,NOTIFY=system_id
//*********************************************************************************************
//* This is sample JCL which can be used to receive an encrypted dataset on a tape and perform   *
//* decryption using the Encryption Facility for z/OS JAVA client. The resulting decrypted data  *
//* is placed into a z/OS dataset.                                                               *
//*                                                                                              *
//* Overview of job steps                                                                        *
//* - STEP1 - this will use IEBGENER to copy an encrypted dataset from a tape and place it       *
//* into a file within the USS file system.                                                      *
//* - STEP2 - this will use the USS file created by STEP1 and invoke BPXBATCH to run a shell     *
//* script named  javadecrypt_pw.sh to decrypt the data into another USS file within a file      *
//* system.                                                                                      *
//* - STEP3 - this will use IEBGENER to copy the decrypted file from the USS file system and     *
//* place it into an z/OS data set.                                                              *
//*                                                                                              *
//* File, dataset and JOB considerations: The originating encrypted dataset is assumed to be     *
//* of Fixed Block (FB) Record Format (RECFM).                                                   *
//*                                                                                              *
//* - All steps within this job specify the REGION= parameter.                                   *
//* This is critical to STEP2 to ensure that JAVA has sufficient memory to execute and decrypt   *
//* the file.                                                                                    *
//*                                                                                              *
//* - The user must ensure that there is sufficient space available within the USS file system   *
//* to contain the encrypted file (which came from tape) and the decrypted file (which was the   *
//* output of the EF JAVA decryption client).                                                    *
//*                                                                                              *
//* - The customer's client (receiver of the encrypted data) must know the data set attributes   *
//* (BLKSIZE,RECFM,LRECL) of the original (clear) source dataset. This information is required   *
//* for STEP3 of this job. It is specified on STEP3 SYSUT1 DD in this sample.                    *
//*                                                                                              *
//* - The customer's client (receiver of the encrypted data) must also know the size of the      *
//* original (clear) source dataset. This information is required for STEP3 of this job. It is   *
//* specified on STEP3, SYSUT2 DD for the SPACE value.                                           *
//*                                                                                              *
//* Note: USS refers to Unix System Services                                                     *
//*                                                                                              *
//*********************************************************************************************
//*********************************************************************************************
//* STEP1 - This jobstep will copy an encrypted file from a tape dataset to an HFS file within   *
//* Unix System Services. The HFS is automatically created via the OCREAT specified on the       *
//* PATHOPTS keyword. The PATHDISP keyword will keep the file active across to the next step.    *
//* Refer to the Unix System Services User's Guide for additional explanation on PATHOPTS,       *
//* PATHMODE and PATHDISP.                                                                       *
//*********************************************************************************************
//STEP1 EXEC PGM=IEBGENER,REGION=6M
//SYSUT1 DD UNIT=3590-1,DSN=HLQ.ENCRYPTED.TAPE.DATASET,
// DISP=(SHR),VOL=SER=ABCDEF
//SYSUT2 DD PATH='/filesys/input.encrypt.file',
// PATHOPTS=(OWRONLY,OCREAT),
// PATHMODE=SIRWXU,
// PATHDISP=KEEP
//SYSPRINT DD SYSOUT=*
//SYSIN DD DUMMY
//*
```

```
//*********************************************************************************************
//* STEP2 - This jobstep will use BPXBATCH to invoke a USS shell script which invokes the EF JAVA *
//* decryption client.The JAVA decryption client uses STDIN as input to specify the USS file      *
//* containing the encrypted data. The JAVA decryption client uses STDOUT as the output USS file  *
//* into which decrypted data will be placed.                                                     *
//*                                                                                               *
//* The shell script is named javadecrypt_pw.sh and the names of the files specified in STDIN and *
//* STDOUT DDs must match the input and output file specifications in the shell script. The PARM  *
//* specified on BPXBATCH assumes that the shell script resides in the USS filesystem in          *
//* a path specified by /path.                                                                    *
//*                                                                                               *
//* STDERR specifies a USS file into which error messages from the JAVA decryption               *
//* client will be placed.                                                                        *
//*                                                                                               *
//* Repeating from the comments above, REGION=0M is critical on STEP2 to ensure that JAVA has     *
//* sufficient storage to decrypt the input file.                                                 *
//*********************************************************************************************
//STEP2 EXEC PGM=BPXBATCH,REGION=0M,
// PARM='SH /path/javadecrypt_pw.sh'
//STDIN DD PATH='/filesys/input.encrypt.file',
// PATHOPTS=(ORDONLY),
// PATHMODE=SIRWXU,
// PATHDISP=KEEP
//STDOUT DD PATH='/filesys/output.decrypt.file',
// PATHOPTS=(OWRONLY,OCREAT),
// PATHMODE=SIRWXU,
// PATHDISP=KEEP
//STDERR DD PATH='/path/stderr.log',
// PATHOPTS=(OWRONLY,OCREAT,OTRUNC),PATHMODE=SIRWXU
//*********************************************************************************************
//* STEP3 - This jobstep will copy a decrypted file from a USS file within the file system to a    *
//* z/OS dataset via IEBGENER. The BLKSIZE, LRECL, and RECFM on SYSUT1 will drive the attributes   *
//* that will be used on SYSUT2 when it is created.                                               *
//*                                                                                               *
//* The space required to handle the decrypted output in the z/OS dataset must be consistent with  *
//* the original (clear) source dataset that came from the originator partner. In this sample,     *
//* the dataset required 300 cylinders on a 3390.                                                 *
//*                                                                                               *
//*********************************************************************************************
//STEP3 EXEC PGM=IEBGENER,REGION=6M
//SYSUT1 DD PATH='/filesys/output.decrypt.file',
// BLKSIZE=6144,LRECL=1024,RECFM=FB,
// PATHOPTS=(ORDONLY),
// PATHMODE=SIRWXU,
// PATHDISP=KEEP
//*
//SYSUT2 DD DSN=HLQ.DECRYPT.OUTPUT,
// DISP=(NEW,CATLG),
// SPACE=(CYL,(2,1)),UNIT=SYSDA
//SYSPRINT DD SYSOUT=*
//SYSIN DD DUMMY
```

Running the Encryption Facility for z/OS Client sample script: `STEP2` of the JCL specifies the shell script `javadecrypt_pw.sh` that invokes Encryption Facility for z/OS Client. The following sample script shows how Encryption Facility for z/OS Client uses the password `TEST1PASSWORD` to decrypt the USS file `/filesys/input.encrypt.file`:

```
# Sample script which may be invoked via z/os Batch process, IE: BPXBATCH
# PATH statement needs to specify the absolute path to the java 'bin' directory.
# CLASSPATH must include the java encryption java client's absolute path and name.
# IE: /path/efclient.zip
# it is important to 'cd' to the directory where the efclient.zip file resides.
# Please note: This example utilizes decryption via PASSWORD.
# This example uses '/filesys/input.encrypt.file' as it's encrypted data source,
# which is also specified in BPXBATCH job as STDIN.
# This example uses '/filesys/output.decrypt.file' for expected decrypted output,
# which is also specified in BPXBATCH job as STDOUT.
# Important: Ensure the 'REGION=0M', parm is included in the BPXBATCH job step,
# to ensure the java virtual machine
# has sufficient storage available.
#
export PATH=/usr/lpp/java/cur13secure/bin:.:$PATH
export CLASSPATH=/feu/efclient.zip:.:$CLASSPATH
cd /path
java -Djava.encryption.facility.debuglevel=0 \
com.ibm.encryptionfacility.EncryptionFacility \
-mode decrypt \
-password TEST1PASSWORD \
-inputFile /filesys/input.encrypt.file \
```

Using the RACDCERT command for key and certificate management of encrypted data

In this scenario you want to create a new certificate with a 2048 bit RSA public/private key pair for an encrypted data set that is sent from a remote site. You plan to send the certificate to the remote site so the recipient can create a PKDS label and store the certificate in the ICSF PKDS on the remote system:

1. Use the RACDCERT command on your site to create the PKDS label for the key pair:

```
RACDCERT GENCERT SUBJECTSDN(CN('Sally''s Data Encryption')) WITHLABEL('Sally''s Data Encryption') SIZE(2048)
  PCICC NOTAFTER(DATE(2020/08/10)
```

2. Use the RACDCERT LIST command to view the name of the PKDS label that RACF has created:

```
RACDCERT LIST(LABEL('Sally''s Data Encryption'))

Digital certificate information for user SALLY:
   Label: Sally's Data Encryption
   Certificate ID: 2QfHxdbZx8XUaqweQMOFmaNA46iXhUBgQOKFmUB7QPDw
   Status: TRUST
   Start Date: 2005/08/11 00:00:00
   End Date:   2020/08/10 23:59:59
   Serial Number:
       >00<
   Issuer's Name:
       >CN=Sally's Data Encryption<
   Subject's Name:
       >CN=Sally's Data Encryption<
   Private Key Type: PCICC
   Private Key Size: 2048
   PKDS Label: IRR.DIGTCERT.SALLY.SY1.BD7103108611F42F
```

3. To send the certificate you need to extract it from RACF. Use the RACDCERT EXPORT command to extract it from RACF:

```
RACDCERT EXPORT(LABEL('Sally''s Data Encryption')) DSN(FOR.JACOB.CRT)
```

4. Send the certificate to the remote site so the recipient can return encrypted data. (You can use e-mail, FTP, or whatever program your installation uses to send the certificate to the remote site. Note that you do not send the private key to the recipient so the certificate does not need to be protected.)
5. The recipient at the remote site receives the certificate that you sent in step 4 into a data set (SALLY.CRT in this example). The recipient needs the public key from this certificate to send you encrypted data. The recipient adds the certificate to the RACF data base as a SITE certificate, and gives it the name “Sally”. The command also creates a PKDS label with the same value “Sally.”

```
RACDCERT SITE ADD(SALLY.CRT) WITHLABEL('Sally') ICSF(*)
```

6. From the remote site the recipient uses the following RACDCERT command to list out the certificate that has been added to RACF. Note that RACF changes PKDS label text to uppercase:

```
RACDCERT SITE LIST(LABEL('Sally'))

Digital certificate information for SITE:
   Label: Sally
   Certificate ID: egljcv8XUaqweQMOFmaNA46iXhUBgQOKFmUB7QPDw
   Status: TRUST
   Start Date: 2005/08/11 00:00:00
   End Date:   2020/08/10 23:59:59
   Serial Number:
       >00<
   Issuer's Name:
       >CN=Sally's Data Encryption<
   Subject's Name:
       >CN=Sally's Data Encryption<
   Private Key Type: None
   PKDS Label: SALLY
```

Using ICSF utilities panels for PKDS key management

This scenario shows how to use the ICSF utilities panels to manage PKDS keys.

1. From the ICSF primary options utility panel, select option 5, UTILITY and press ENTER. You receive the ICSF utilities panel CSFUTL00:

```
------------------------------ ICSF - Utilities --------------------------

 Enter the number of the desired option.

   1  ENCODE     -  Encode data
   2  DECODE     -  Decode data
   3  RANDOM     -  Generate a random number
   4  CHECKSUM   -  Generate a checksum and verification and
                    hash pattern
   5  PPKEYS     -  Generate master key values from a pass phrase
   6  PKDSKEYS   -  Manage keys in the PKDS









 Press ENTER to go to the selected option.
 Press END   to exit to the previous menu.


 OPTION ===>
```

2. From CSFUTL00 select option 6: PKDSKEYS and press enter. You receive the ICSF PKDS keys panel CSFPKY00.

```
------------------------------ ICSF - PKDS Keys -------------------------
    Enter the PKDS record's label for the actions below
     ==>

    Select one of the following actions then press ENTER to process:

    _  Generate a new PKDS key pair record
       Enter the key length  ===>          512, 1024, or 2048
       Enter Private Key Name (optional)
        ==>

    _  Delete the existing public key or key pair PKDS record

    _  Export the PKDS record's public key to a certificate data set
       Enter the DSN  ===>
       Enter desired subject's common name (optional)
        CN=

    _  Create a PKDS public key record from an input certificate.
       Enter the DSN  ===>


 COMMAND ===>
```

CSFPKY00 allows you to perform the following PKDS key management:

- Generate an RSA key pair PKDS record
- Delete an existing PKDS record
- Export an existing public key to an x.509 certificate
- Import a public key from an x.509 certificate

Coprocessor Requirements for using the ICSF utility panels: To use the full function of the ICSF utility panels, you must have a PCICC, PCIXCC, or a CEX2C cryptographic coprocessor. If you do not have one of these coprocessors, you cannot generate key pairs using the panels.

For example, to generate a new PKDS RSA key pair, enter the PKDS record label (MY.PKDS.KEY.PAIR) and the key length in bits. You can also specify an optional private key name that ICSF can imbed into the key token. This service creates an RSA public/private key PKDS record. You can use the key pair that ICSF generates to encrypt and recover archive data. You can also use it to recover encrypted data that another party transmits to you if you make the public key portion available to the partner:

```
------------------------------ ICSF - PKDS Keys -------------------------
    Enter the PKDS record's label for the actions below
     ==>MY.PKDS.KEY.PAIR

    Select one of the following actions then press ENTER to process:

    _  Generate a new PKDS key pair record
       Enter the key length  ===> 1024       512, 1024, or 2048
       Enter Private Key Name (optional)
        ==>

    _  Delete the existing public key or key pair PKDS record

    _  Export the PKDS record's public key to a certificate data set
       Enter the DSN  ===>
       Enter desired subject's common name (optional)
        CN=

    _  Create a PKDS public key record from an input certificate.
       Enter the DSN  ===>


 COMMAND ===>
```

If the system successfully generates the new key pair record, you receive confirmation panel CSFPKY01:

```
---------------------- ICSF - PKDS Key Request Successful ---------------


   Label ==> MY.PKDS.KEY.PAIR




   Key function completed successfully

   Press ENTER or END to return to the previous menu.









COMMAND ===>
```

For complete information about using the ICSF utilities panels, see *z/OS Cryptographic Services ICSF Administrator's Guide*.

Appendix. Notices

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

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

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

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

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

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

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

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

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

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

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

IBM Corporation
Department number/Building number_
Site mailing address_
City, State; Zip Code_
U.S.A. (or appropriate country)

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

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

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

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

COPYRIGHT LICENSE:

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

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

Programming interface information

This book is a user's guide to Encryption Facility.

This book primarily documents information that is NOT intended to be used as a Programming Interface of Encryption Facility.

However, this information may also contain diagnosis, modification, and tuning information. Diagnosis, modification and tuning information is provided to help you debug your application software.

Caution: Do not use this diagnosis, modification, and tuning information as a programming interface because it is subject to change.

Trademarks

The following terms used in this publication are trademarks of the IBM Corporation in the United States and/or other countries:

DFSMSdfp™
DFSMSdss
DFSMShsm
IBM
MVS
OS/390
RACF
Redbooks
iSeries
xSeries
z/OS
z/Series

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

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

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

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

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

Index

Numerics

C

D

E

H

I

J

K

L

M

P

R

S

T

U

V

Readers' Comments — We'd Like to Hear from You

Encryption Facility for z/OS:
User's Guide
Version 1 Release 1.0

Publication No. SA23-1349-03

Overall, how satisfied are you with the information in this book?

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Overall satisfaction	☐	☐	☐	☐	☐

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

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Accurate	☐	☐	☐	☐	☐
Complete	☐	☐	☐	☐	☐
Easy to find	☐	☐	☐	☐	☐
Easy to understand	☐	☐	☐	☐	☐
Well organized	☐	☐	☐	☐	☐
Applicable to your tasks	☐	☐	☐	☐	☐

Please tell us how we can improve this book:

Thank you for your responses. May we contact you? ☐ Yes ☐ No

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you. IBM or any other organizations will only use the personal information that you supply to contact you about the issues that you state on this form.

Name

Address

Company or Organization

Phone No.

Readers' Comments — We'd Like to Hear from You
SA23-1349-03

IBM

Cut or Fold
Along Line

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

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

IBM Corporation
Department 55JA, Mail Station P384
2455 South Road
Poughkeepsie, NY
12601-5400

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

Cut or Fold
Along Line

Program Number: 5655-P97

Printed in USA

SA23-1349-03