IBM Security Key Lifecycle Manager for z/OS
Version 1.1

Planning, and User's Guide

SC14-7628-00

IBM Security Key Lifecycle Manager for z/OS
Version 1.1

Planning, and User's Guide

SC14-7628-00

Note
Before using this information and the product it supports, read the information in "Notices" on page 153.

April 2011

This edition applies to version 1, release 1 of IBM Security Key Lifecycle Manager for z/OS (product number 5698-B35) and to all subsequent releases and modifications.

Contents

Figures

Tables

About this Publication

This book contains information to help you install, configure, and use IBM® Security Key Lifecycle Manager for z/OS. It includes concepts and procedures pertaining to:

- Encryption on the IBM System Storage® TS1120, TS1130, TS1140, DS8000® Turbo drive, IBM LTO Ultrium 4 Tape Drive and IBM LTO Ultrium 5 Tape Drive.
- Cryptographic keys
- Digital certificates

Intended Audience

This book is intended for storage and security administrators responsible for security and backup of vital data, and anyone assisting in the setup and maintenance of Security Key Lifecycle Manager for z/OS servers in the operating environment. It assumes the reader has a working knowledge of storage devices and networks.

Publications

Read the descriptions of the product library and the related publications to determine which publications you might find helpful. After you determine the publications you need, see the instructions for accessing publications online.

Related Publications

The following publications provide information related to encryption on tape drives:

IBM System Storage TS1120 and TS1130 Tape Drive and Controller Publications

- *IBM System Storage TS1120 Tape Drive and Controller Operator Guide*, GA32-0556
- *IBM System Storage TS1120 and TS1130 Tape Drives and TS1120 Controller Introduction and Planning Guide*, GA32-0555-04
- *IBM System Storage TS1120 Tape Drive SCSI Reference*, GA32-0562
- *IBM TotalStorage Enterprise Silo Compatible Tape Frame 3592 Introduction, Planning, and User's Guide*, GA32-0463

IBM LTO Ultrium 4 Tape Drive Publications

- *IBM LTO Ultrium 4 Tape Drive Setup, Operator, and Service Guide*, GA27-2102

IBM System Storage TS3500 Tape Library Publications

- *IBM System Storage TS3500 Tape Library Operator Guide*, GA32-0560
- *IBM System Storage TS3500 Tape Library Introduction and Planning Guide*, GA32-0559

IBM Virtualization Engine TS7700 Publications

- *IBM Virtualization Engine TS7700 Series Introduction and Planning Guide*, GA32-0567

IBM 3953 Tape System Publications

- *IBM 3953 Library Manager Model L05 Operator Guide*, GA32-0558
- *IBM 3953 Tape System Introduction and Planning Guide*, GA32-0557

IBM TotalStorage Enterprise Automated Tape Library (3494) Publications

- *IBM TotalStorage Automated Tape Library (3494) Introduction and Planning Guide*, GA32-0448
- *IBM TotalStorage Automated Tape Library (3494) Operator's Guide*, GA32-0449

zSeries—S390 Publications

- *IBM eServer™ zSeries 900 Platform Reference Guide*, G326-3092
- *Introduction to IBM S/390® FICON*, SG24-5176 (IBM Redbook).
- *S/390 System Overview Parallel Enterprise Server — Generation 5*, GA22-7158
- *S/390 System Overview Parallel Enterprise Server — Generation 6*, GA22-1030

IBM Fibre Channel Publications

- *IBM TotalStorage SAN Switch 2109 Model F16 Installation and Service Guide*, SY27-7623
- *IBM Fiber-Optic Channel Link Planning and Installation*, GA32-0367

IBM FICON Publications

- *FICON (FCV Mode) Planning Guide*, SG24-5445-00 (IBM Redbook).
- *Planning for: Fiber Optic Links (ESCON®, FICON, Coupling Links, and Open system Adapters)*, GA23-0367
- *Maintenance Information for: Fiber Optic Links (ESCON, FICON, Coupling Links, and Open System Adapters)*, SY27-2597
- *Fiber Channel Connection (FICON) I/O Interface Physical Layer*, SA24-7172
- *Introduction to IBM System/390® FICON*, SG24-5176
- *Planning for the ED-5000 Enterprise Fibre Channel Director*
- *IBM eServer zSeries Connectivity Handbook*, SG24-5444
- *IBM Tape Solutions for Storage Area Networks and FICON*, SG24-5474

Related Software Publications

For information regarding software related to the IBM 3592 Tape System, refer to:

- *IBM Tape Device Drivers Installation and User's Guide*, GC35-0154
- *Basic Tape Library Support User's Guide and Reference*, SC26-7016
- *Environmental Record Editing and Printing (EREP) Program User's Guide and Reference*, GC35-0151
- *IBM Tivoli Storage Manager for AIX® Administrator's Guide*, GC32-0768
- *z/OS DFSMS Introduction*, SC26-7397
- *z/OS DFSMS Object Access Method Planning, Installation, and Storage Administration Guide for Tape Libraries*, SC35-0427
- *z/OS DFSMS Software Support for IBM System Storage TS1130 and TS1120 Tape Drives (3592)*, SC26-7514
- *z/OS Migration*, GA22-7499.
- *z/OS Security Server RACF Security Administrator's Guide*, SA22-7683
- *z/OS Security Server RACF Command Language Reference*, SA22-7687
- *z/OS Cryptographic Services Integrated Cryptographic Service Facility System Programmer's Guide*, SA22-7520

- *z/OS Cryptographic Services Integrated Cryptographic Service Facility Administrator's Guide*, SA22-7521
- *z/OS Cryptographic Services Integrated Cryptographic Service Facility Application Programmer's Guide*, SA22-7522
- *z/OS Cryptographic Services System Secure Sockets Layer Programming* , SC24-5901
- *z/VM General Information Version 4 Release 3.0*, GC24-5991
- *z/VM CP Planning and Administration*, SC24-6083
- *z/VM CP Commands and Utilities*, SC24-6081
- *IBM eServer™ zSeries® 900 Platform Reference Guide*, G326-3092
- *Introduction to IBM S/390® FICON®*, SG24-5176 (IBM Redbook) G326-3092
- *S/390 System Overview Parallel Enterprise Server — Generation 5*, GA22-7158
- *S/390 System Overview Parallel Enterprise Server — Generation 6*, GA22-1030

Other Publications

- *American National Standard Institute Small Computer System Interface* X3T9.2/86-109 X3.180, X3B5/91-173C, X3B5/91-305, X3.131-199X Revision 10H, and X3T9.9/91-11 Revision 1

Accessibility

Accessibility features help users with a physical disability, such as restricted mobility or limited vision, to use software products successfully. The publications for this product are in Adobe Portable Document Format (PDF) and should be compliant with accessibility standards. Use assistive technologies, such as screen-reader software and digital speech synthesizer, to hear what is displayed on the document. Consult the product documentation of the assistive technology for details on using those technologies with this product.

Accessing publications online

The publications for this product are available online in Portable Document Format (PDF) or Hypertext Markup Language (HTML) format, or both in the Tivoli® software library.

The Tivoli software library is located at http://publib.boulder.ibm.com/tividd/td/tdprodlist.html.

To locate product publications in the library, click the first letter of the product name or scroll until you find the product name. Click the product name. Product publications can include release notes, installation guides, user's guides, administrator's guides, and developer's references.

IBM posts publications for this and all other Tivoli products, as they become available and whenever they are updated, to the Tivoli Documentation Central Web site at http://www.ibm.com/tivoli/documentation.

Note: To ensure proper printing of PDF publications, select the **Fit to page** check box in the Adobe Acrobat Print window (which is available when you click **File** > **Print**).

You can also locate publications at http://www.elink.ibmlink.ibm.com/publications/servlet/pbi.wss.

IBM Java Security Components and Keystores

- Overview technical article: **http://www.ibm.com/developerworks/library/j-ibmsecurity.html**
- SDK 5.0 : **http://www.ibm.com/developerworks/java/jdk/security/50/**
- SDK 6.0 : **http://www.ibm.com/developerworks/java/jdk/security/60/**
- hwkeytool: **http://www.ibm.com/servers/eserver/zseries/software/java/hwkeytool.html**
- ikeyman: For ikeyman and the keytool user guide: SDK 5.0 or SDK 6.0 links above
- For additional information specific to z/OS®: **http://www.ibm.com/servers/eserver/zseries/software/java/**

IBM Storage Media Support

The following URL provides access to current regional and country-specific IBM addresses and telephone numbers.

- **http://www.storage.ibm.com/media/distributors**

IBM TotalStorage Enterprise Tape System 3592 Support

For general information about the 3592 Tape System, visit the following URL:

- **http://www.ibm.com/servers/storage/tape/3592/index.html**

For general information about the TS1120 Tape Drive, visit the following URL:

- **http://www.ibm.com/servers/storage/tape/ts1120/index.html**

For information about supported servers for the 3592 Tape System and TS1120 tape Drive, visit the following URL:

- **http://www.ibm.com/servers/storage/tape/compatibility/pdf/3592_interop.pdf**

The following URLs provide access to additional current information related to 3592 Tape System.

Device Driver Support: To access the 3592 Firmware and Device Driver Matrix, visit the following URL:

- **http://www.ibm.com/servers/storage/support/tape/3592/downloading.html**

To access the TS1120 Firmware and Device Driver Matrix, visit the following URL:

- **http://www.ibm.com/servers/storage/support/tape/ts1120/downloading.html**

You can download device driver software and read documentation about various device drivers at the following URL:

- **ftp://ftp.software.ibm.com/storage/devdrvr/**

IBM Virtualization Engine TS7700 Encryption Support: White paper: *IBM Virtualization Engine TS7700 Series Encryption Overview* available at

- **http://www.ibm.com/support/docview.wss?&uid=ssg1S4000504**

IBM Network Integration and Deployment Services: The following URL provides information about connectivity and the integration of cabling systems.

- **http://www.ibm.com/services/networking/integration**

IBM Tape Storage Publications: Use this URL for IBM Hardware product documents in a PDF format for viewing and printing.

- **http://www.ibm.com/servers/storage/tape/resource-library.html#publications**

SAN Fabric: This link provides information on high-performance switches and gateways.

- **http://www-03.ibm.com/systems/storage/product/**

I/O Connectivity: This link provides updated information regarding FICON® and fibre channel connectivity.

- **http://www.ibm.com/servers/eserver/zseries/connectivity**

Redbooks: Use this URL to access the IBM Redbooks®:

- **http://www.redbooks.ibm.com/**

Vendor Support: These URLs provide compatibility information in PDF format for implementing software, servers, and operating systems with IBM tape drives and libraries.

- For TS1120: **http://www.ibm.com/servers/storage/tape/compatibility/pdf/ts1120_isv_matrix.pdf**
- For 3592: **http://www.ibm.com/servers/storage/tape/compatibility/pdf/3592_isv_matrix.pdf**

Support information

If you have a problem with your IBM software, you want to resolve it quickly. IBM provides the following ways for you to obtain the support you need:

Online

Access the IBM Software Support site at http://www.ibm.com/software/support/probsub.html .

Conventions used in this publication

This publication uses several conventions for special terms and actions, operating system-dependent commands and paths, and margin graphics.

Typeface conventions

This information uses these typeface conventions.

Bold

- Lowercase commands and mixed case commands that are otherwise difficult to distinguish from surrounding text
- Interface controls (check boxes, push buttons, radio buttons, spin buttons, fields, folders, icons, list boxes, items inside list boxes, multicolumn lists, containers, menu choices, menu names, tabs, property sheets), labels (such as **Tip:**, and **Operating system considerations**:)
- Keywords and parameters in text

Italic

- Citations (examples: titles of publications, diskettes, and CDs
- Words defined in text (example: a nonswitched line is called a *point-to-point line*)
- Emphasis of words and letters (words as words example: "Use the word *that* to introduce a restrictive clause."; letters as letters example: "The LUN address must start with the letter *L*.")

- New terms in text (except in a definition list): a *view* is a frame in a workspace that contains data.
- Variables and values you must provide: ... where *myname* represents....

Monospace

- Examples and code examples
- File names, programming keywords, and other elements that are difficult to distinguish from surrounding text
- Message text and prompts addressed to the user
- Text that the user must type
- Values for arguments or command options

Chapter 1. Product Overview

The Product Overview topic explains the capabilities of the product and how it can be used to protect data.

Data is one of the most highly valued resources in a competitive business environment. Protecting that data, controlling access to it, and verifying its authenticity while maintaining its availability are priorities in our security-conscious world. Data encryption is a tool that answers many of these needs. The Security Key Lifecycle Manager for z/OS simplifies encryption tasks.

The Security Key Lifecycle Manager for z/OS supports several storage devices. The IBM System Storage TS1130 Tape Drive (3592 Model E06 and Model EU61. The 3592 EU6 Tape Drive is a 3592 E05 Tape Drive canister upgraded to contain a Model E06 drive through the MES (Miscellaneous Equipment Specification) process) and TS1120 Tape Drive (3592 Model E05) can encrypt data as it is written to any size IBM TotalStorage Enterprise Tape Cartridge 3592, including WORM cartridges. The IBM System Storage TS1140 Tape Drive (3592 Model E07) can encrypt data on IBM TotalStorage Enterprise Tape Cartridge 3592 JB/JX and JC/JY media only in supported write format. JA/JW cartridges can only read encrypted or decrypted data. The IBM LTO Ultrium 4 Tape Drive (LTO Ultrium 4) and LTO Ultrium 5 Tape Drive (LTO Ultrium 5) drives can also encrypt data as it is written to any LTO Ultrium 4 and LTO Ultrium 5 Data Cartridges. Encryption is performed at full line speed in the tape drive after compression. Compression is more efficiently done before encryption. This new capability adds a strong measure of security to stored data. It is added without the processing overhead and performance degradation associated with encryption performed on the server or the expense of a dedicated appliance.

The encryption solution comprises these elements:

The Encryption-Enabled Devices

All TS1130 and TS1140 Tape Drives are encryption-capable. All TS1120 Tape Drives with Feature Code 5592 or 9592 are *encryption-capable*. All IBM LTO Ultrium 4 and LTO Ultrium 5 Tape Drives are *encryption-capable*. Fibre-Channel (FC) and Serial Attached SCSI (SAS) IBM LTO-4 and LTO-5 tape drives are encryption-capable. SCSI IBM LTO-4 and LTO-5 tape drives are encryption aware, can load and handle encrypted LTO-4 and LTO-5 cartridges, but cannot process encryption operations. These tape drives are functionally capable of performing hardware encryption, but this capability has not yet been activated. In order to perform hardware encryption, the tape drives must be *encryption-enabled*.

In an IBM System Storage TS3500 Tape Library, the TS1120, TS1130, and TS1140 tape drives can be encryption-enabled through the IBM System Storage Tape Specialist.

Note: When a TS1130 Tape Drive is attached to a 3592 J70 or C06 tape controller, the tape drive must be enabled for system-managed encryption. This setting applies even when encryption is not being used by the host.
For all other TS1120, TS1130, and TS1140 tape drives, an IBM representative sets up the drive for encryption. Only encryption-enabled TS1120, TS1130, and TS1140 tape drives can read and write encrypted 3592 tape cartridges.

All IBM LTO Ultrium 4 and LTO Ultrium 5 Tape Drives can be encryption-enabled through the IBM System Storage Tape Specialist. However, encryption must be licensed on LTO Ultrium 4 and LTO Ultrium 5 Tape Drives in tape libraries. This setting is acquired with Feature Code 1604 on the TS3500 Library or Feature Code 5900 on other libraries. Consult your library documentation for more information.

The IBM® System Storage® DS8000® with IBM Full Disk Encryption drives are encryption-capable. Each storage facility image on an encryption-capable DS8000 can be configured to either enable or disable encryption for all data that is stored on your disks. To enable encryption, the DS8000 must be configured to communicate with two or more Security Key Lifecycle Manager for z/OS servers. The physical connection between the DS8000 HMC and the key server is through a TCP/IP network. To learn additional information about the DS8000, see http://publib.boulder.ibm.com/infocenter/dsichelp/ds8000ic/index.jsp?topic=%2Fcom.ibm.storage.ssic.help.doc%2Ff2c_ekmtklm_3ekm3r.html

See "Hardware and Software Requirements" on page 17 for more information about tape drives.

Encryption Key Management
Encryption involves the use of several kinds of keys, in successive layers. The generation, maintenance, control, and transmission of these keys depends upon the operating environment where the encrypting tape drive is installed. Some applications such as Tivoli Storage Manager, can do key management. For environments without such applications or where application-agnostic encryption is wanted, IBM provides the Security Key Lifecycle Manager for z/OS to perform all necessary key management tasks. "Managing Encryption" on page 8 describes these tasks in more detail.

Encryption Policy
This policy is used to implement encryption. It includes the rules that govern which volumes are encrypted and the mechanism for key selection. How and where these rules are set up depends on the operating environment. See "Managing Encryption" on page 8 for more information.

Note: In the Tape Storage environment, the Encryption function on tape drives is configured and managed by the customer and not the IBM System Services Representative (SSR). In some instances SSRs are required to enable encryption at a hardware level when service access or service password controlled access is required. Customer setup support is by Field Technical Sales Specialist (FTSS), customer documentation, and software support for encryption software problems. Customer "how to" support is also provided on the support line contract.

Security Key Lifecycle Manager for z/OS Components

Provides information on the various components of the software and their function.

The Security Key Lifecycle Manager for z/OS is part of the IBM Java environment and uses the IBM Java Security components for its cryptographic capabilities. (For more information about the IBM Java Security components see the related publications section.) The Security Key Lifecycle Manager for z/OS has four main components that are used to control its behavior. These components are:

Java security keystore
The keystore is defined as part of the Java Cryptography Extension (JCE).

The keystore is an element of the Java Security components, which are, in turn, part of the Java runtime environment. A keystore holds the certificates and keys (or pointers to the certificates and keys) used by the Security Key Lifecycle Manager for z/OS to perform cryptographic operations. Several types of Java keystores are supported offering different operational characteristics to meet your needs. These characteristics are explained in detail in "Keystore Considerations" on page 26.

Attention: It is impossible to overstate the importance of preserving your keystore data. You will not be able to decrypt your encrypted tapes if you do not have access to your keystore. Carefully read the relevant topics to understand the methods available for protecting your keystore data.

Configuration files
The configuration files enable you to customize the behavior of the Security Key Lifecycle Manager for z/OS to meet the needs of your organization.

Device Table
The device table is used by the Security Key Lifecycle Manager for z/OS to monitor the devices it supports. The device table is a non-editable, binary file whose location is specified in the configuration file. You can change its location to meet your needs.

KeyGroups.xml file
This password-protected file contains the names of all encryption key groups and the aliases of the encryption keys associated with each key group.

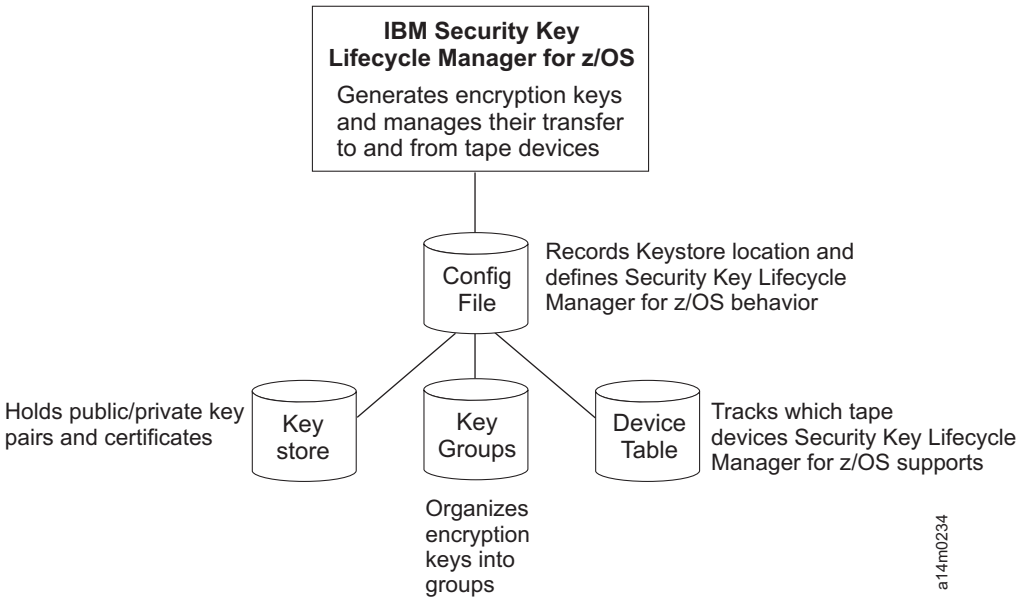

Figure 1. The four main components of Security Key Lifecycle Manager for z/OS

Technical overview

You can use Security Key Lifecycle Manager for z/OS to assist in securing vital data.

The Security Key Lifecycle Manager for z/OS works with IBM encryption-enabled tape drives and system storage devices. The product helps in generating, protecting, storing, and maintaining encryption keys that are used to encrypt information being written to and decrypt information being read from devices.

Encryption-enabled 3592 and LTO tape drives

Security Key Lifecycle Manager for z/OS supports encryption-enabled 3592 and LTO tape drives. Drives without encryption enablement are not supported.

Security Key Lifecycle Manager for z/OS supports these drive types:

- 3592 tape drives

  TS1120, TS1130, and TS1140 tape drives that are enabled to encrypt data.
- LTO

  LTO Ultrium 4 and LTO Ultrium 5 tape drives that are enabled to encrypt data.

Encryption is performed at full line speed in the tape drive after compression.

Enterprise Storage - IBM System Storage DS8000 (2107, 242x)

Security Key Lifecycle Manager for z/OS supports the DS8000 Storage Controller.

This support requires the appropriate microcode bundle version on the DS8000 Storage Controller, Licensed Internal Code level 64.2.xxx.0 or higher.

Keys overview

An encryption key is typically a random string of bits generated specifically to scramble and unscramble data. Encryption keys are created using algorithms designed to ensure that each key is unique and unpredictable. The longer the key constructed this way, the harder it is to break the encryption code.

Federal Information Processing Standard 140-2 Considerations

Describes the Federal Information Processing Standard (FIPS) 140-2 cryptographic standard and its relation to the product.

Federal Information Processing Standard (FIPS) 140-2 has become important now that the Federal government requires all its cryptographic providers to be FIPS 140 certified. This standard has also been adopted in a growing private sector community. The certification of cryptographic capabilities by a third-party in accordance with government standards is felt to have increased value in this security-conscious world.

The Security Key Lifecycle Manager for z/OS does not provide cryptographic capabilities itself and therefore does not require, nor is it allowed to obtain, FIPS 140-2 certification. However, the Security Key Lifecycle Manager for z/OS takes advantage of the cryptographic capabilities of the IBM JVM in the IBM Java Cryptographic Extension component. The product allows the selection and use of the IBMJCEFIPS cryptographic provider, which has a FIPS 140-2 level 1 certification. Turn the **fips** configuration parameter to **on** in the Configuration Properties file. This setting makes the Security Key Lifecycle Manager for z/OS use the IBMJCEFIPS provider for all cryptographic functions.

Note: Hardware-based keystore types cannot be used with the FIPS parameter set to on. Only JCEKS and JCERACFKS can be used.

You can find more information about the IBMJCEFIPS provider and its selection and use at http://www.ibm.com/developerworks/java/jdk/security/50/FIPShowto.html.

See the documentation from specific hardware and software cryptographic providers for information about whether their products are FIPS 140-2 certified.

About Encryption Keys

This topic describes the encryption keys used to encrypt data.

An encryption key is typically a random string of bits generated specifically to scramble and unscramble data. Encryption keys are created using algorithms designed to ensure that each key is unique and unpredictable. The longer the key constructed this way, the harder it is to break the encryption code. Both the IBM and T10 methods of encryption use 256-bit AES algorithm keys to encrypt data. 256-bit AES is the encryption standard currently recognized by the U.S. government, which allows three different key lengths. 256-bit keys are the longest allowed by AES.

Two types of encryption algorithms can be used by the Security Key Lifecycle Manager for z/OS: symmetric algorithms and asymmetric algorithms. Symmetric, or secret key encryption, uses a single key for both encryption and decryption. Symmetric key encryption is generally used for encrypting large amounts of data efficiently. 256-bit AES keys are symmetric keys. Asymmetric, or public and private encryption, uses a pair of keys. Data encrypted using one key can only be decrypted using the other key in the public and private key pair. When an asymmetric key pair is generated, the public key is typically used to encrypt, and the private key is typically used to decrypt.

The Security Key Lifecycle Manager for z/OS uses both symmetric and asymmetric keys. Symmetric encryption is used for high-speed encryption of user or host data, and asymmetric encryption (which is necessarily slower) for protecting the symmetric key.

Encryption keys can be generated by the Security Key Lifecycle Manager for z/OS, by applications such as Tivoli Storage Manager, or by a utility such as keytool. Generating AES keys and how they are transferred to the tape drive depend on the tape drive type and the method of encryption management. It is helpful to understand the difference between how the Security Key Lifecycle Manager for z/OS uses encryption keys and how other applications use them.

How the Security Key Lifecycle Manager for z/OS Processes Encryption Keys

How Security Key Lifecycle Manager for z/OS encrypts data depends on the type of drive.

- DS8000

  Encrypts data using IBM® Full Disk Encryption drives.
- TS1120, TS1130, and TS1140 Tape Drives

  Converts data to ciphertext using a symmetric 256-bit AES Data Key.
- LTO Ultrium 4 Tape Drive and LTO Ultrium 5 Tape Drive

  Converts data to ciphertext using a pre-generated symmetric Data Key.

On DS8000

Security Key Lifecycle Manager for z/OS supports data encryption with the IBM® Full Disk Encryption drives for DS8000. To use data encryption, a DS8000 must be ordered from the factory with all IBM Full Disk Encryption drives.

For more information about DS8000 encryption, see http://publib.boulder.ibm.com/infocenter/dsichelp/ds8000ic/index.jsp?topic=%2Fcom.ibm.storage.ssic.help.doc%2Ff2c_ds8000encryption_3ekm6r.html

On TS1120, TS1130, and TS1140 Tape Drives

In system-managed and library-managed tape encryption, unencrypted data (clear text) is sent to the tape drive. The text is then converted to ciphertext using a symmetric 256-bit AES Data Key (DK) generated by the Security Key Lifecycle Manager for z/OS. The ciphertext is then written to tape. The Security Key Lifecycle Manager for z/OS uses a single, unique Data Key for each Enterprise Tape Cartridge. This Data Key is also encrypted, or wrapped, by the Security Key Lifecycle Manager for z/OS using the public key from an asymmetric Key Encrypting Key (KEK) pair. This process creates an Externally Encrypted Data Key (EEDK). The EEDK is written to the cartridge memory and to three additional places on the tape media in the cartridge. The tape cartridge now holds both the encrypted data and the means to decrypt it for anyone holding the private part of the KEK pair. Figure 2 illustrates this process.

The DK is also wrapped a second time, possibly using the public key of another party, to create an additional EEDK. Both EEDKs can be stored on the tape cartridge. In this way, the tape cartridge can be shipped to a business partner holding the corresponding private key. The private key would allow the DK to be unwrapped and the tape decrypted by the business partner.

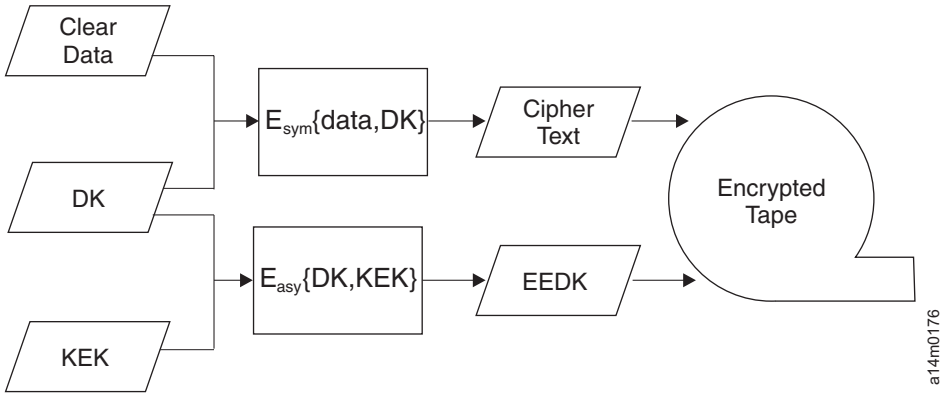

Figure 2. Encryption Using both Symmetric and Asymmetric Encryption Keys. System-Managed and Library-Managed Encryption on TS1120TS1130, and TS1140 tape drives

On the LTO Ultrium 4 Tape Drive and LTO Ultrium 5 Tape Drive

In system-managed and library-managed tape encryption, unencrypted data is sent to an LTO Ultrium 4 or LTO Ultrium 5 Tape Drive. The data is then converted to ciphertext using a pre-generated symmetric Data Key (DK) from a keystore available to the Security Key Lifecycle Manager for z/OS. The data is then written to tape. The Security Key Lifecycle Manager for z/OS selects a pre-generated Data Key in round robin fashion. Data Keys are reused on multiple tape cartridges when an insufficient number of Data Keys have been pre-generated. The Data Key is sent to the LTO Ultrium 4 or LTO Ultrium 5 Tape Drive in encrypted, or wrapped, form by the Security Key Lifecycle Manager for z/OS. The LTO Ultrium

4 or LTO Ultrium 5 Tape Drive unwraps this Data Key and uses it to perform encryption or decryption. However, no wrapped key is stored anywhere on the LTO Ultrium 4 or LTO Ultrium 5 tape cartridge. This method is a major difference between the way TS1120, TS1130, or TS1140 and LTO devices operate with the Security Key Lifecycle Manager for z/OS. Once the encrypted volume is written, the Data Key must be accessible based on the alias or key label. The Data key must be available to the Security Key Lifecycle Manager for z/OS in order for the volume to be read. Figure 3 illustrates this process.

The Security Key Lifecycle Manager for z/OS also gives you the ability to organize your symmetric keys for LTO encryption into key groups. This feature allows you to group keys according to the type of data they encrypt, or by any other specification. See "Creating and managing key groups" on page 76 for more information.

Encryption Key Processing by Other Applications (Security Key Lifecycle Manager for z/OS not Used)

In application-managed tape encryption in which Security Key Lifecycle Manager for z/OS is not used, unencrypted data (clear text) is sent to the tape drive. The text is then converted to ciphertext using a symmetric Data Key (DK) provided by the application, and is then written to tape. The Data Key is not stored anywhere on the tape cartridge. When the encrypted volume is written, the Data Key must be in a location available to the application for the volume to be read.

TS1120, TS1130, TS1140, LTO Ultrium 4, and LTO Ultrium 5 tape drives can use applications such as Tivoli Storage Manager for application-managed encryption. Tivoli Storage Manager uses a single, unique Data Key for each tape cartridge.

Alternatively, the tape drives can be used by applications that use the T10 command set to perform encryption. The T10 command set uses symmetric 256-bit AES keys provided by the application. T10 can use multiple, unique Data Keys per tape cartridge, and even write encrypted data and clear data to the same tape cartridge. When the application encrypts a tape cartridge, it selects or generates a Data Key. The Data key is generated using a method determined by the application and sends it to the tape drive. The key is **not** wrapped with an asymmetric public key and it is **not** stored on the tape cartridge. When the encrypted data is written to tape, the Data Key must be in a location available to the application. This setting enables data to be read.

The process for application-managed tape encryption (and system-managed and library-managed encryption on LTO) is shown in Figure 3.

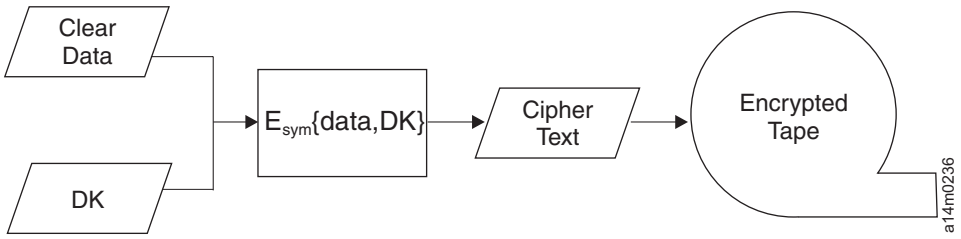

Figure 3. Encryption Using only Symmetric Encryption Keys. Application-Managed Encryption on TS1120, TS1130, and TS1140 tape drives, and System-Managed, Library-Managed, and Application-Managed Encryption on LTO Ultrium 4 and LTO Ultrium 5 Tape Drive.

In Summary

The number of encryption keys that can be used for each volume depends on the:

- Tape drive
- Encryption standard
- Method used to manage the encryption

For transparent encryption of LTO Ultrium 4 and LTO Ultrium 5 (that is, using system-managed or library-managed encryption with the Security Key Lifecycle Manager for z/OS,) the uniqueness of Data Keys depends on the availability of a sufficient number of pre-generated keys to the Security Key Lifecycle Manager for z/OS.

Table 1. Encryption Key Summary

Encryption Management Method	Keys used by		
	TS1120 TS1130, and TS1140 (IBM Encryption)	**LTO Ultrium 4 and LTO Ultrium 5 (IBM Encryption)**	**TS1120, TS1130, TS1140/LTO Ultrium 4 and LTO Ultrium 5 (T10 Encryption)**
System-Managed Encryption / Library-Managed Encryption (Security Key Lifecycle Manager for z/OS)	1 unique DK / cartridge	1 DK / cartridge	N/A
Application-Managed Encryption (no Security Key Lifecycle Manager for z/OS)	1 unique DK / cartridge	1 DK / cartridge	Multiple DKs / cartridge
DK = Symmetric AES 256-bit Data Key			

Managing Encryption

The Security Key Lifecycle Manager for z/OS is a Java software program. This product assists IBM encryption-enabled devices. The product assists in generating, protecting, storing, and maintaining encryption keys. Those keys are used to encrypt information being written to, and decrypt information being read from, tape media (tape and cartridge formats) and system storage devices. This product is designed to run in the background as a shared resource deployed in several locations within an enterprise. The product can serve numerous IBM encrypting tape drives and system storage devices regardless of where those devices are. These devices can be in tape library subsystems, connected to mainframe systems through various types of channel connections, or installed in other computing systems. A command-line interface client provides a robust set of commands to customize the Security Key Lifecycle Manager for z/OS for your environment and monitor its operation. The product uses one or more keystores to hold the certificates and keys (or pointers to the certificates and keys). These keystores are required for all encryption tasks. The Security Key Lifecycle Manager for z/OS supports the following IBM keystores:

- JCEKS
- JCECCAKS
- JCECCARACFKS
- JCERACFKS

See “Keystore Considerations” on page 26 for detailed information.

Attention: The product performs the function of requesting the generation of encryption keys and passing those keys to TS1120, TS1130, TS1140 or LTO Ultrium 4,LTO Ultrium 5 tape drives and DS8000. The key material, in wrapped (encrypted) form resides in system memory during processing by the Security Key Lifecycle Manager for z/OS. The key material must be transferred without error to the appropriate tape drive so that data written on a cartridge can be recovered (decrypted). If a corrupted key material is used to write data to a cartridge, then data written to that cartridge will not be recovered. There are safeguards in place to make sure that such data errors do not occur. If the machine hosting the Security Key Lifecycle Manager for z/OS is not using Error Correction Code (ECC) memory it is possible that the key material can become corrupted while in system memory. The corruption can then cause data loss. Although the risk is slight, it is best to host the Security Key Lifecycle Manager for z/OS using ECC memory.

The Security Key Lifecycle Manager for z/OS acts as a background process. The product waits for key generation or key retrieval requests. The requests are sent to it through a TCP/IP communication path between itself and the tape library, tape controller, tape subsystem, device driver, or tape drive. When a tape drive writes encrypted data, it first requests an encryption key from the Security Key Lifecycle Manager for z/OS. Upon receipt of the request, the Security Key Lifecycle Manager for z/OS performs the following tasks.

For TS1120, TS1130, and TS1140 tape drives: The Security Key Lifecycle Manager for z/OS generates an Advanced Encryption Standard (AES) key and serves it to the tape drives in two protected forms:

- Encrypted or *wrapped*, using Rivest-Shamir-Adleman (RSA) key pairs. TS1120, TS1130, and TS1140 tape drives write this copy of the key to the cartridge memory. The key is also copied in three additional places on the tape media in the cartridge for redundancy.
- Separately wrapped for secure transfer to the tape drive. The keys are unwrapped upon arrival, and the key inside is used to encrypt the data written to tape.

When an encrypted tape cartridge is read by a supported tape drive, the protected AES key on the tape is sent to the Security Key Lifecycle Manager for z/OS. The wrapped AES key is unwrapped in the Security Key Lifecycle Manager for z/OS. The AES key is then wrapped with a different key for secure transfer back to the tape drive. The AES key is unwrapped in the tape drive and used to decrypt the data stored on the tape. The Security Key Lifecycle Manager for z/OS also allows protected AES keys to be rewrapped, or *rekeyed*. Different RSA keys can be used from the originals when the tape was written. Rekeying is useful when an unexpected need arises to export volumes to business partners whose public keys were not included. This method eliminates rewriting the entire tape. This method also enables the data key of a tape cartridge to be reencrypted with the public key of the business partner.

For LTO Ultrium 4 Tape Drive and LTO Ultrium 5 Tape Drive: The Security Key Lifecycle Manager for z/OS fetches an existing AES key from a keystore. The product then wraps the AES key for secure transfer to the tape drive where it is unwrapped upon arrival. The key is used to encrypt the data being written to tape.

When an encrypted tape is read by an LTO Ultrium 4 or LTO Ultrium 5 Tape Drive, the Security Key Lifecycle Manager for z/OS fetches the required key from the keystore. The key is fetched based on the information in the Key ID on the tape and serves it to the tape drive wrapped for secure transfer.

For DS8000: When the DS8000 starts, the device requests an unlock key from Security Key Lifecycle Manager for z/OS.

If the DS8000 requests a new key for its unlock key, Security Key Lifecycle Manager for z/OS generates an Advanced Encryption Standard (AES) key and serves the key to the drive in two protected forms:

- Encrypted (wrapped) using Rivest-Shamir-Adleman (RSA) key pairs. The DS8000 stores this copy of the key on the array in an unencrypted partition.
- Separately wrapped for secure transfer to the drive where it is unwrapped upon arrival and the key inside is used to unlock the array.

If the DS8000 requests an existing unlock key, the protected AES key on the array is sent to Security Key Lifecycle Manager for z/OS where the wrapped AES key is unwrapped. The AES key is then wrapped with a different key for secure transfer back to the DS8000, where it is unwrapped and used to unlock the array.

For TS1120, TS1130,TS1140, LTO Ultrium 4 Tape Drive and LTO Ultrium 5 Tape Drive: There are three methods of encryption management to choose from. These methods have the following differences:

- Where the encryption policy engine resides
- Where key management is performed for your solution
- How the Security Key Lifecycle Manager for z/OS is connected to the drive

Your operating environment determines which is the best for you. Key management and the encryption policy engine can be located in any one of the following three environmental layers.

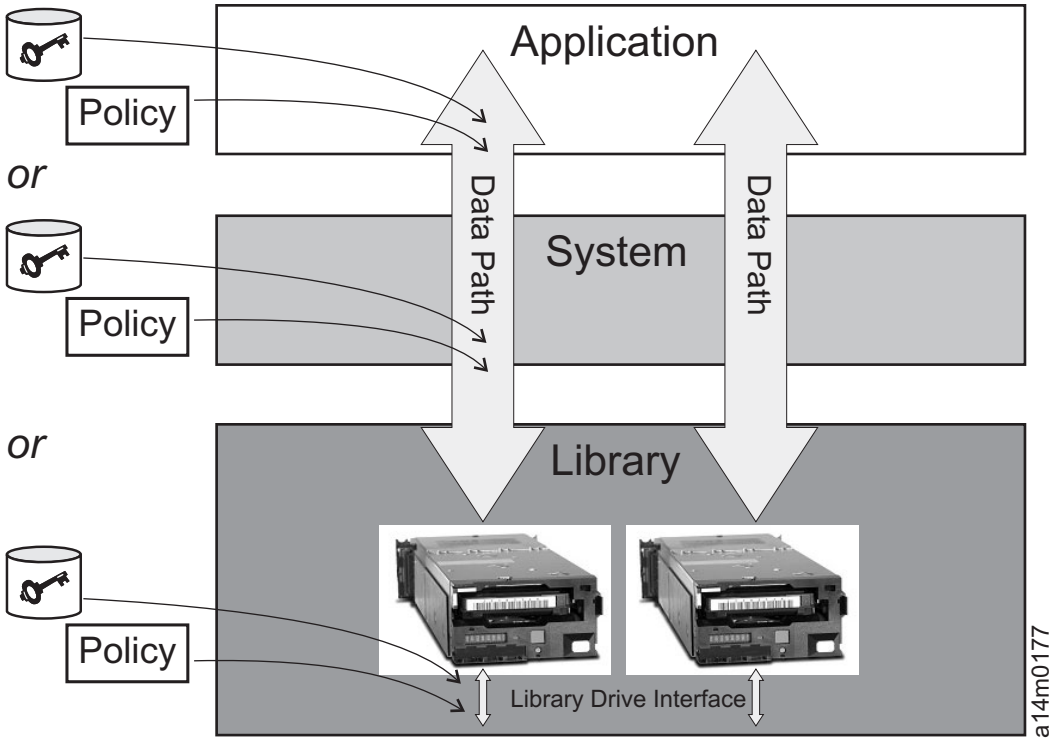

Figure 4. Three possible locations for encryption policy engine and key management.

Application Layer
An application program, separate from the software, initiates data transfer for tape storage, for example Tivoli Storage Manager.

System Layer
Everything between the application and the tape drives, for example the operating system, z/OS DFSMS, device drivers, and FICON/ESCON controllers.

Library Layer
The enclosure for tape storage, such as the IBM System Storage TS3500 Tape Library. A modern tape library contains an internal interface to each tape drive within it.

Application-Managed Tape Encryption

This topic describes application-managed tape encryption.

This method is best where operating environments run an application already capable of generating and managing encryption policies and keys, such as Tivoli Storage Manager. Policies specifying when encryption is to be used are defined through the application interface. The policies and keys pass through the data path between the application layer and the encrypting tape drives. Encryption is the result of interaction between the application and the encryption-enabled tape drive, and does not require any changes to the system and library layers. The application manages the encryption keys. Because of this setup, data volumes written and encrypted using the application-managed encryption method can only be read by the same software application that wrote them.

The Security Key Lifecycle Manager for z/OS is not required by, or used by, application-managed tape encryption.

Application-managed tape encryption on IBM TS1120, TS1130,TS1140, LTO Ultrium 4, and LTO Ultrium 5 tape drives can use either of two encryption command sets:

- The IBM encryption command set developed for the Security Key Lifecycle Manager for z/OS
- The T10 command set defined by the InterNational Committee for Information Technology Standards (INCITS)

Application-managed tape encryption using the TS1120, and TS1130 tape drives are supported in the following IBM libraries:

- IBM System Storage TS3400 Tape Library
- IBM System Storage TS3500 Tape Library
- IBM TotalStorage 3494 Tape Library

Application-managed tape encryption using the TS1140 tape drive is supported in the IBM System Storage TS3500 Tape Library.

Application-managed tape encryption using LTO Ultrium 4 and LTO Ultrium 5 tape drives are supported in the following IBM tape drives and libraries:

- IBM System Storage TS2340 Tape Drive Express® Model S43 and via Xcc/HVEC 3580S4X
- IBM System Storage TS3100 Tape Library
- IBM System Storage TS3200 Tape Library
- IBM System Storage TS3310 Tape Library
- IBM System Storage TS3500 Tape Library

For details about setting up Application-Managed tape encryption, see your Tivoli Storage Manager documentation or visit http://publib.boulder.ibm.com/infocenter/tivihelp/v1r1/index.jsp for more information.

System-Managed Tape Encryption

This topic describes system-managed tape encryption.

Use this method for encryption on TS1120, TS1130, TS1140, LTO Ultrium 4, and LTO Ultrium 5 tape drives. Use this method if the applications that write or read from tape are not capable of performing the key management required for application-managed encryption.

System z®

Encryption policies specifying when to use encryption are set up in z/OS DFSMS (Data Facility Storage Management Subsystem). They can also be done implicitly through each instance of IBM device driver. Additional software products such as IBM Integrated Cryptographic Service Facility (ICSF) and IBM Resource Access Control Facility (RACF®) can also be used. Key generation and management are performed by the Security Key Lifecycle Manager for z/OS, a Java application running on the host or externally on another host. Policy controls and keys pass through the data path between the system layer and the encrypting tape drives. Encryption is transparent to the applications.

Encryption key labels are assigned on a per-storage pool basis. The labels are assigned using the TS7700 Maintenance Interface on the following tape drives connected to an IBM Virtualization Engine TS7700:

- TS1120
- TS1130
- TS1140

DFSMS storage constructs are used by z/OS to control the use of storage pools for logical volumes, resulting in an indirect form of encryption policy management. For more information, see the white paper, *IBM Virtualization Engine TS7700 Series Encryption Overview*, available at http://www.ibm.com/support/docview.wss?&uid=ssg1S4000504.

With system-managed encryption, System z hosts can rekey an encrypted tape on the TS1120, TS1130, and TS1140 tape drives. See the appropriate operating system documentation for the mechanism that initiates a rekey operation. For example, with z/OS, the existing IEHINITT utility is enhanced to support rekeying. Use rekeying to export volumes to multiple business partners. This method eliminates rewriting the entire tape and enables the data key of a tape cartridge to be reencrypted with the public key of the business partner.

See *z/OS DFSMS Software Support for IBM System Storage TS1130 and TS1120 Tape Drives (3592)* for more details on setting up system-managed encryption.

Encryption Key Paths

In system-managed encryption on System z platforms, multiple key paths are supported by the tape controller.

In-Band Key Flow

In-band key flow occurs between the Security Key Lifecycle Manager for z/OS and the tape drive, shown in Figure 5. The key flow occurs through a FICON proxy on the FICON/ESCON interface. The FICON proxy supports failover to the secondary key path if first-specified Security Key Lifecycle Manager for z/OS path addresses fail. Impact on controller service requirements is minimal.

The controller

- Reports drive status in SMIT displays
- Passes encryption-related errors from the drive to the host
- Reports to the host "encryption failure unit checks"
- Must be reconfigured whenever new encryption drives are introduced for attachment or when an encryption-capable drive is enabled for encryption.

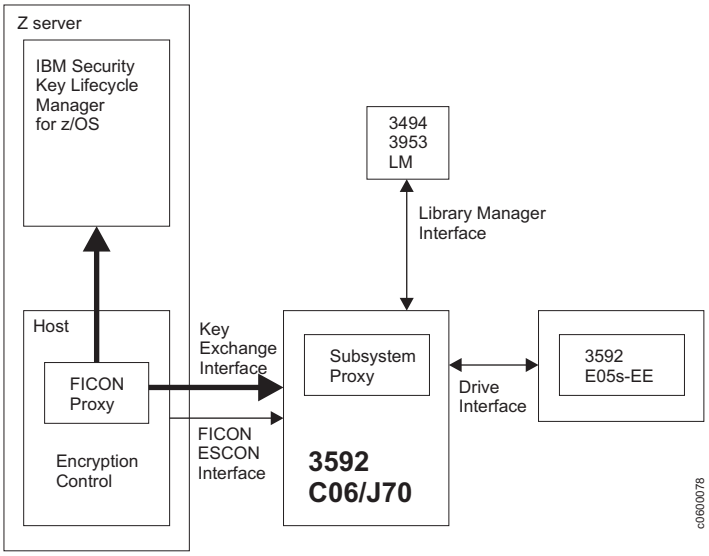

Figure 5. In-band encryption key flow

Out-of-Band Key Flow

Out-of-band key flow, shown in Figure 6 on page 14, occurs between the Security Key Lifecycle Manager for z/OS and the tape drive. The key flow occurs through a subsystem proxy, located in the 3592 Controller or TS7700 Virtualization Engine, on the Security Key Lifecycle Manager for z/OS interface. Impact on service requirements can be greater than for in-band key flow. The impact is greater due to the introduction of two routers on the Security Key Lifecycle Manager for z/OS interface, to and from the controller.

The controller and TS7700

- Supports failover to the secondary key path on failure of first-specified Security Key Lifecycle Manager for z/OS path addresses
- Reports drive status in SMIT displays
- Passes encryption-related errors from the drive to the host
- Reports to the host "encryption failure unit checks"
- Must be reconfigured whenever new encryption drives are introduced for attachment or when an encryption-capable drive is enabled for encryption.

As many as two Security Key Lifecycle Manager for z/OS IP/domain addresses (and as many as two ports) can be entered for each controller. Two Domain name server IP addresses can also be entered.

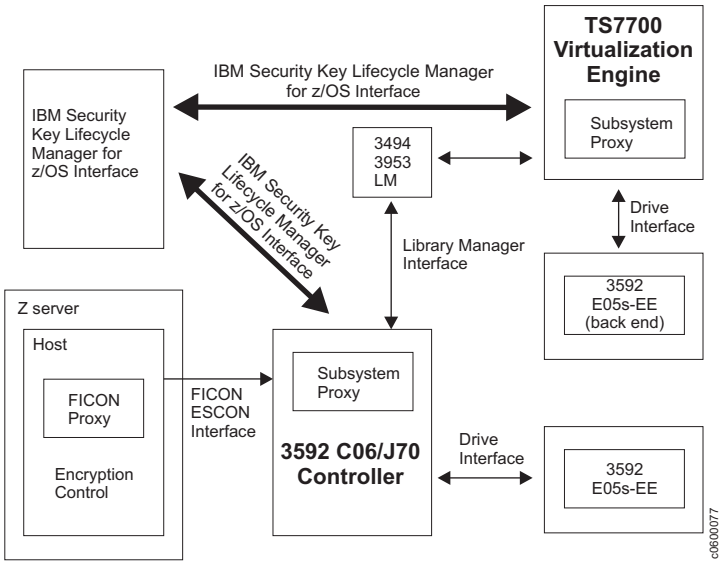

Figure 6. Out-of-band encryption key flow

Library-Managed Tape Encryption

This topic describes library-managed tape encryption.

This method is best for TS1120, TS1130, TS1140,LTO Ultrium 4 Tape Drive, and LTO Ultrium 5 Tape Drive in the either of the following environments:

- an open-attached IBM System Storage TS3100, TS3200, TS3310, TS3400 or TS3500 tape library, or
- IBM TotalStorage 3494 Tape Library

For TS3500, barcode encryption policies can be used. The policies are used to specify when to use encryption and are set up through the IBM System Storage Tape Library Specialist Web interface. In such cases, policies are based on cartridge volume serial numbers. Library-managed encryption also enables other options, such as encryption of all volumes in a library, independent of bar codes. Key generation and management are performed by the Security Key Lifecycle Manager for z/OS, a Java application running on a library network-attached host. Policy control and keys pass through the library-to-drive interface, therefore encryption is transparent to the applications.

Library-managed encryption, when used with certain applications such as Symantec Netbackup™ or the EMC Legato NetWorker, includes support for an *internal label option*. When the internal label option is configured, either of the following tape drives automatically derive the encryption policy:

- TS1120
- TS1130
- TS1140
- LTO Ultrium 4 Tape Drive
- LTO Ultrium 5 Tape Drive

The tape drives also derive the key information from the metadata written on the tape volume by the application. See your *Tape Library Operator's Guide* for more information.

System-managed tape encryption and library-managed tape encryption interoperate with one another. A tape encrypted using system-managed encryption can be decrypted using library-managed encryption, and vice versa, provided they both have access to the same keys and certificates.

Audit Records

Security Key Lifecycle Manager for z/OS provides audit records in the audit log. The Security Key Lifecycle Manager for z/OS provides System Management Facilities support for audit records.

System Management Facilities is a z/OS service aid for collecting information from various z/OS subsystems. The default configuration on z/OS routes all audit records to System Management Facilities type 83 sub-type 6 records. For more information on request and response formats, see http://www-01.ibm.com/support/docview.wss?uid=pub1sa22763009.

Chapter 2. Planning your Security Key Lifecycle Manager for z/OS Environment

The Planning topic explains how best to use and configure the product.

This section is intended to provide information to allow you to determine the best Security Key Lifecycle Manager for z/OS configuration for your needs. Many factors must be considered when you are planning how to set up your encryption strategy. Review these topics with care.

Hardware and Software Requirements

Security Key Lifecycle Manager for z/OS has specific hardware and software requirements for its installation and use.

Java requirements

You need to install a Java SDK. Understand the requirements needed to use the product correctly.

The minimum Java levels required to run the Security Key Lifecycle Manager for z/OS on the z/OS platform are

- Java SDK 5.0 Service Refresh (SR) 5
- Java SDK 6.0 GA+

Note: Only the IBM version of the Java Runtime Environment (JRE) for each of the following platforms supports the Security Key Lifecycle Manager for z/OS.

Table 2. Security Key Lifecycle Manager for z/OS Minimum Software Requirements for z/OS

IBM Software Developer Kit	Model/PID Number
IBM 31 - bit and 64 - bit SDK for z/OS, Java 2 Technology, V5.0 (z/OS 1.10, 1.11 and 1.12)	5655-N98 (at the SDK 5.0 SR5 level or above)
IBM 31 - bit and 64 - bit SDK for z/OS, Java 2 Technology, V6.0 (z/OS 1.10, 1.11 and 1.12)	5655-R31 for 31 - bit and 5655-R32 for 64 - bit
Available at : http://www.ibm.com/servers/eserver/zseries/software/java	

Note: In Java 6.0 for 64-bit SDK, the ICSF level must be HCR7770 when you use the JCECCAKS keystore type. If the ICSF version is not updated to HCR7770, and the flag `requiredHardwareProtectionForSymmetricKeys` is set to `true`, an error occurs during read and write actions for TS1120, TS1130, and TS1140 tape drives and DS8000.

z/OS Solution Components

Operating Systems

Security Key Lifecycle Manager for z/OS supports z/OS 1.10 ,1.11 and 1.12 versions.

Running Security Key Lifecycle Manager for z/OS

The Security Key Lifecycle Manager for z/OS is a product installed by SMP/E.

Note: Regardless of which IBM SDK version you use, you must replace the **US_export_policy.jar** and **local_policy.jar** files from the **$JAVA_HOME/lib/security** directory. You must replace the files with an unrestricted version of these files. These unrestricted policy files are required by the Security Key Lifecycle Manager for z/OS in order to serve AES keys. The preferred method to replace the files on z/OS is to copy the files that are shipped in the z/OS Java SDK build. The files are under the jce demo directory. Copy the files to the **lib/security** directory:

```
cp /usr/lpp/java/J5.0/demo/jce/policy-files/unrestricted/*
/usr/lpp/java/J5.0/lib/security
```

Alternatively, the unrestricted policy files can be downloaded from the following website: https://www14.software.ibm.com/webapp/iwm/web/preLogin.do?source=jcesdk. Select the unrestricted JCE policy files for the SDK you are currently using. See https://www14.software.ibm.com/webapp/iwm/web/preLogin.do?source=jcesdk for the correct unrestricted policy files.

Control Units

Table 3. Control Unit Requirements for z/OS

Control Unit	Model/PID Number	Type of Update
3592 J70	3592-J70	Firmware update shipped with 3592-J70 FC 5595 and FC 9595
TS1120 C06	3592-C06	Firmware update shipped 3592-C06 FC 5595 and FC 9595
Router for Security Key Lifecycle Manager for z/OS Attach (only required on tape controllers for out-of-band support)		FC 5593 or FC 9593
3494 Lxx	3494 Lxx	Firmware shipped with 3592-J70 or C06 FC 5595 and FC 9595
3953 L05	3953-L05	Firmware shipped with 3592-J70 or C06 FC 5595 and FC 9595

Virtualization Engine TS7700

TS7700 Feature Code 9900 is required for encryption. TS7700 Feature Code 0521 provides the latest firmware updates.

Table 4. Virtualization Engine TS7700 Connected Library Requirements for z/OS

Tape Library	Model/PID Number	Type of Update
TS3500	3584-L22, L23, D22, D23	For firmware update, visit http://www.ibm.com/servers/storage/support/lto/3584/downloading.html
3494	3494-D22	Firmware update shipped with FC 5596 and FC 9596

Tape Libraries

Table 5. Tape Library Requirements for z/OS

Tape Library	Model/PID Number	Type of Update
TS3500	3584-L22, L23, D22, D23	For firmware update, visit http://www.ibm.com/servers/storage/support/lto/3584/downloading.html
3494	3494-L22, D22, D24	Firmware update shipped with FC 5595 and FC 9595

Tape Drive

Table 6. Tape Drive Requirements for z/OS

Tape Drive	Model/PID Number	Type of Update
TS1120	3592-E05 New drive order	Order 3592-E05 FC9592 and 3592-J70 or C06 FC9595 or FC5595
	3592-E05 Field upgrade for encryption	
TS1130	3592-E06	All E06 and EU6 drives are encryption capable. Order FC9596 for the E06 or EU6 to configure for encryption-enablement.
	3592-EU6	
TS1140	3592-E07	All E06 and EU6 drives are encryption capable. Order FC9596 for the E06 or EU6 to configure for encryption-enablement.

z/VM Solution Components

Operating Systems

z/VM 5.2 and later, plus DFSMS/VM FL221 if running z/VSE® guests.

PTFs for the following APARs are required:

z/VM 5.2.0 - VM64063
z/VM 5.2.0 and later - VM64459
DFSMS/VM FL221 - VM64062 & VM64458

Note: The Security Key Lifecycle Manager for z/OS does not run on z/VM®. However, the product is supported through an out-of-band connection to a Security Key Lifecycle Manager for z/OS server running on z/OS. See "Planning for System-Managed Tape Encryption" on page 21. See "z/OS Solution Components" on page 17 for z/OS prerequisites.

See the latest editions of the following publications for more information:

- *z/VM CP Planning and Administration* (SC24-6083) - Chapter 23 describes the overall usage of encryption support on z/VM.
- *z/VM CP Commands and Utilities* (SC24-6081) - Contains specifics about each command.

Encryption Setup Tasks at a Glance

Before you can use the encryption capability of the tape drive, you must be sure that certain software and hardware requirements are met. The following checklists are intended to help you meet these requirements.

Note: Contact your IBM Representative for additional information about encryption on either of the following tape drives: IBM TS1120, TS1130,TS1140, LTO Ultrium 4, or LTO Ultrium 5 Tape Drive.

Security Key Lifecycle for z/OS Manager Setup Tasks

Familiarize yourself with the required configuration tasks to enable the product to communicate with the supported drives.

Before you can encrypt tapes, the Security Key Lifecycle Manager for z/OS must first be configured and running so that it can communicate with the encrypting tape drives. The Security Key Lifecycle Manager for z/OS is not required to run while tape drives are being installed. However, the product must be running in order to perform encryption.

The following information lists the tasks you must perform before using the Security Key Lifecycle Manager for z/OS.

- Decide what system platform to use as Security Key Lifecycle Manager for z/OS server.
- Upgrade server operating system if necessary. (“Hardware and Software Requirements” on page 17.)
- Upgrade the Java Virtual Machine if necessary. (“Hardware and Software Requirements” on page 17.)
- Install Java Unrestricted Policy Files. (“Hardware and Software Requirements” on page 17.)
- Install the Security Key Lifecycle Manager for z/OS as instructed in the Program Directory document. See, Program Directory for IBM Security Key Lifecycle Manager for z/OS.
- Decide on keystore type. (“Which Keystore is Right for You” on page 36.)
- Create keys, certificates, and key groups.

  “Example 1: Using the Java Keytool and JCEKS on z/OS” on page 48

  “Example 2: Using the JCECCAKS Keystore with the Java Hwkeytool on z/OS” on page 50 (z/OS only)

  “Example 3: Using the JCERACFKS or JCECCARACFKS Keystore on z/OS” on page 53 (z/OS only)

  “Generating Keys and Aliases for Encryption on LTO Ultrium 4 and LTO Ultrium 5” on page 72

  “Creating and managing key groups” on page 76
- If necessary, import keys and certificates.
- Define the configuration properties file. (Chapter 4, “Configuring the Security Key Lifecycle Manager for z/OS,” on page 79.)
- Define tape drives to the Security Key Lifecycle Manager for z/OS or set **drive.acceptUnknownDrives** configuration property value on. Similarly for DS8000 devices, set **ds8k.acceptUnknownDrives** to on or define these devices. (See “adddrive” on page 92 to define drives explicitly, or see “Automatically update device table” on page 79.)

- Start the Security Key Lifecycle Manager for z/OS server. (To start the Security Key Lifecycle Manager for z/OS see, "Quick Test Running Security Key Lifecycle Manager for z/OS Under USS" on page 66.)

Planning for Application-Managed Tape Encryption

This topic provides the setup tasks required for application-manage tape encryption.

To perform encryption, the TS1120, TS1130, TS1140, LTO Ultrium 4 Tape Drive, or LTO Ultrium 5 Tape Drive must be encryption-capable.

Application-Managed Tape Encryption Setup Tasks

Any task not identified below as an IBM service task is the responsibility of the customer.

1. Install and cable either of the following tape drives: TS1120, TS1130, TS1140 (IBM service task), LTO Ultrium 4 or LTO Ultrium 5.
   a. Update library firmware (3494, TS3500 where applicable)
   b. Update tape drive firmware (all tape drives in same library or environment)
2. Enable encryption for either of the following drives: TS1120, TS1130, TS1140, LTO Ultrium 4, or LTO Ultrium 5. See *IBM System Storage TS3500 Operator's Guide* for configuring TS1120, TS1130, TS1140, LTO Ultrium 4 and LTO Ultrium 5 on TS3500. Configuring the TS1120, TS1130, or TS1140 tape drives on all others is an IBM service task.
3. Install appropriate IBM tape device driver level (Atape, for example) where required by application.
4. Set up encryption policies. See *IBM Tivoli Storage Manager for z/OS Administrator's Guide*.
5. Perform write/read operation to test encryption.
6. Verify encryption of the test volume by Autonomic Management Engine (AME): issue

   ```
   QUERY VOLUME FORMAT=DETAILED
   ```

   Verify that `Drive Encryption Key Manager` is set to `Tivoli Storage Manager`.

Planning for System-Managed Tape Encryption

This topic presents the required setup tasks for system-managed tape encryption.

The requirements to perform system-managed encryption are:

- Encryption-capable TS1120, TS1130, TS1140, LTO Ultrium 4, or LTO Ultrium 5 tape drive.
- Keys and corresponding certificates.
- Security Key Lifecycle Manager for z/OS component for the Java platform.
- Routers and cables for out-of-band Security Key Lifecycle Manager for z/OS-to-TS1120, TS1130, or TS1140 tape drive path (System z platforms only).

Setup Tasks for System-Managed Tape Encryption on IBM System z Platforms

Any task not identified below as an IBM service task is the responsibility of the customer.

1. Install and cable either of the following tape drives: TS1120, TS1130, or TS1140 tape drive (IBM service task).
   a. Update tape drive firmware (3592 Models E05, J1A in same environment)
   b. Update 3494, TS3500, and 3953 tape system library firmware (System z platforms or 3953 in heterogeneous environment)
   c. Update 3592 Models C06, J70 Tape Controller firmware (System z platforms or tape controllers in heterogeneous environment) (optional)
   d. Update TS7700 Virtualization Engine microcode.
2. Encryption-enable either of the following tape drives: TS1120, TS1130, or TS1140 tape drive. Refer to *IBM System Storage TS3500 Operator's Guide* for configuring either of the following tape drives on TS3500: TS1120, TS1130, or TS1140. The 3494 Web Specialist can now be used to enable encryption on either of the following drives in a 3494 Tape Library: TS1120, TS1130, or TS1140. See *IBM TotalStorage Automated Tape Library (3494) Operator's Guide*. For TS7700-attached drives, specify the system-managed encryption method. The configuration of other tapes is an IBM service task.
3. Install tape controller code update, Feature Code 5595 (IBM service task).
4. Install, cable, and configure routers to the Security Key Lifecycle Manager for z/OS, Feature Code 5593 (for out-of-band path to the Security Key Lifecycle Manager for z/OSonly). This is an IBM service task.
   - Define Primary/Secondary Security Key Lifecycle Manager for z/OS IP ports for the tape controller.
5. Update z/OS and DFSMS host software with appropriate PTFs.
6. Install Feature Code 9900 License Key on TS7700.
7. Set up encryption policies.
   - Update DFSMS Data Class to specify encryption (recording format EE2) and other optional parameters (such as media type or performance scaling) as appropriate.
   - Specify the key labels through the DD statement, data class, or Security Key Lifecycle Manager for z/OS defaults.
   - Update other DFSMS polices (as appropriate) to steer allocation to correct library.
   - Encryption on the TS7700 VE is controlled on a storage pool basis. Use the Maintenance Interface (MI) web interface for the TS7700 VE Pool Encryption Settings panel to specify the key labels and modes to use for each storage pool.

   See *IBM z/OS DFSMS Software Support for IBM System Storage TS1130 and TS1120 Tape Drives (3592)*.
8. For in-band key management use the IECIOSxx PARMLIB member or **SETIOS** command to define Primary/Secondary Security Key Lifecycle Manager for z/OS. Also define the IOSAS OMVS segment to RACF.
9. Make the appropriate HCD changes.
10. Determine if coexistence support is needed.
11. Contact your tape management system or application vendor for any required code changes and any installation exit changes that are needed.
12. Set up the system-managed encryption method. For 3494 or stand-alone drives, have your IBM service representative update the drives. For TS3500, update using the IBM System Storage Tape Library Specialist.
13. Schedule an IPL.
14. Verify encryption:

For in-band path to the Security Key Lifecycle Manager for z/OS:

a. Use the `DISPLAY IOS,ISKLM` command (with the VERIFY option) to verify the in-band path to the Security Key Lifecycle Manager for z/OS.
b. Verify that a job (or application) requesting encryption (through data class) has its data encrypted.

For out-of-band path to the Security Key Lifecycle Manager for z/OS:

a. Use RAS functions to verify (IBM service task) Security Key Lifecycle Manager for z/OS paths and encryption configuration.

Planning for Library-Managed Tape Encryption

This topic provides the required setup tasks for library-managed encryption.

The requirements to perform system-managed encryption are:

- Encryption-capable TS1120, TS1130, TS1140, LTO Ultrium 4, or LTO Ultrium 5 tape drive
- Keystore
- Security Key Lifecycle Manager for z/OS component for the Java platform

Library-Managed Tape Encryption Tasks

Any task not identified below as an IBM service task is the responsibility of the customer.

1. Install and cable either of the following tape drives: TS1120, TS1130, TS1140 (IBM service task), LTO Ultrium 4, or LTO Ultrium 5 tape drive.
   a. Update tape system library firmware (3494 or TS3500)
   b. Update tape drive firmware (all tape drives in same library)
   c. For TS1120, TS1130, or TS1140 tape drives in 3494 or TS3500, order Feature Code 9900 for Encryption Configuration. This is an IBM service task.
   d. For LTO Ultrium 4 tape drives, install Feature Code 1604 for Transparent LTO Encryption. This is an IBM service task.
2. Perform the following steps:
   - Use IBM System Storage Tape Library Specialist to enable TS1120, TS1130, TS1140, LTO Ultrium 4, or LTO Ultrium 5 tape drives.
   - Use IBM System Storage Tape Library Specialist to enable 3494 or TS3500 Tape Library for library-managed tape encryption. See the appropriate tape library operator guide.

   a. Add Security Key Lifecycle Manager for z/OS IP addresses.
   b. Specify key label.
   c. Set up scratch encryption policy.
3. Set up key mapping for ILEP (optional).
4. Use library diagnostic functions to verify Security Key Lifecycle Manager for z/OS paths and encryption configuration.

TS1120, TS1130, and TS1140 Tape Drive Installation Process for Encryption

Before the IBM service representative does the installation or upgrade on the TS1120 (3592 Model E05), TS1130 (3592 Model E06), or TS1140 (3592 Model E07) Tape Drives for encryption, you must:

- Decide which method of encryption management to use (application-managed encryption, system-managed encryption, or library-managed encryption). See "Managing Encryption" on page 8.
- Install and configure the Security Key Lifecycle Manager for z/OS. See Chapter 3, "Installing the Security Key Lifecycle Manager for z/OS and Keystores," on page 43.

Encryption Setup Procedure for IBM Service

About this task

The following steps are performed by the IBM Service Representative:

Procedure

1. Record the serial numbers of all 3592 E05, E06, or EU6 tape drives. Provide these numbers to the customer (optional if the customer plans to set the Security Key Lifecycle Manager for z/OS configuration to drive.acceptUnknownDrives=true for automatic addition of tape drives to device table).
2. Install 3592 E05, E06, EU6 Tape Drives
   a. If adding new 3592 E05, E06, or EU6 encryption-capable drives to an existing frame, see *3494 Maintenance Information* or *3584 Maintenance Information* for installation instructions. When installation is complete, continue at step 3.
   b. If replacing current 3592 E05, E06, or EU6 drives with new ones, see *3494 Maintenance Information* or *3584 Maintenance Information* for drive FRU replacement instructions. When replacement is complete, continue at step 3.
   c. If upgrading current 3592 E05, E06, or EU6 drives, see *Feature Code 5592 MES Installation Instructions*. When upgrade is complete, continue at step 3.
3. Configure 3592 E05, E06, or EU6 tape drives for Encryption.
   a. If 3592 E05, E06, or EU6 tape drives are installed in an Enterprise System and connected to a 3592 C06 or J70, you must use system-managed encryption only. If the drives are installed in a 3494 or Standalone Frame, go to step 4. If the drives are installed in a 3584 library, configure and encryption-enable the tape drives using the System Storage Tape Specialist web interface. When tape drives are configured, continue at step 5.
   b. If 3592 E05, E06, or EU6 tape drives are installed in a 3494 Open System, the 3494 Web Specialist can now be used to enable-encryption them. See *IBM TotalStorage Automated Tape Library (3494) Operator's Guide*. Then continue at step 5.
   c. If 3592 E05, E06, or EU6 tape drives are installed in a 3584 Open System, the customer configures, and encryption-enables the tape drives using the System Storage Tape Specialist web interface. When tape drives are configured, continue at step 5.
4. Encryption-enable the 3592 tape drives by following the procedure for "Setting Drive Encryption" in *3592 Maintenance Information*. When tape drives are encryption-enabled, continue at step 6.
5. Use the System Storage Tape Specialist web interface to verify that the 3592 Tape Drives are encryption-enabled. For example, select **Manage Library** > **By Logical Library** > (Select Library) **Modify Encryption Method** > **GO**.
6. If the 3592 E05, E06, or EU6 tape drives are encryption-enabled for Enterprise Systems, follow the 3592 C06 or J70 *Maintenance Information* and *Installation and*

Configuration Guide (ICG). Follow the guide to configure the controllers for encryption, then continue at step 7. If the 3592 E05, E06, or EU6 tape drives are installed in a rack, continue at step 8.

7. Run Library Verify.
8. Go to End-Of-Call Procedures in the appropriate *Maintenance Information*.

LTO Ultrium 4 Tape Drive and LTO Ultrium 5 Tape Drive Installation Process for Encryption

Familiarize yourself with the required installation process for the supported LTO drives.

- Decide which method of encryption management to use (application-managed encryption, system-managed encryption, or library-managed encryption). See "Managing Encryption" on page 8.
- Install and configure the Security Key Lifecycle Manager for z/OS. SeeChapter 3, "Installing the Security Key Lifecycle Manager for z/OS and Keystores," on page 43.
- Use the System Storage Tape Specialist web interface to verify that the LTO Ultrium 4 and LTO Ultrium 5 tape drives are encryption-enabled. Select **Manage Library** > **By Logical Library** > (Select Library) **Modify Encryption Method** > GO.

See "Configuring Security Key Lifecycle Manager for z/OS for LTO Ultrium 4 and LTO Ultrium 5 encryption" on page 87 for more information about LTO.

DS8000 Installation Process for Encryption

The following describes the installation process for IBM System Storage DS8000 for encryption.

- Install and configure the Security Key Lifecycle Manager for z/OS. See Chapter 3, "Installing the Security Key Lifecycle Manager for z/OS and Keystores," on page 43.
- Ensure that the DS8000 is encryption-enabled. For more information about DS8000 encryption, see http://publib.boulder.ibm.com/infocenter/dsichelp/ds8000ic/index.jsp?topic=%2Fcom.ibm.storage.ssic.help.doc%2Ff2c_enc_bestpractice_intro_3ekm9r.html
- Create or import certificates for the Security Key Lifecycle Manager for z/OS keystore, using the corresponding keytool.
- Define the DS8000 to the Security Key Lifecycle Manager for z/OS or set **ds8k.acceptUnknownDrives** configuration property value to `true`. (See "adddrive" on page 92 to define drives explicitly, or see "Automatically update device table" on page 79.)

  Note: The certificate configured in the DS8000 is required so that Security Key Lifecycle Manager for z/OS can automatically add the DS8000.

For more information about installation information related to your DS8000, see http://publib.boulder.ibm.com/infocenter/dsichelp/ds8000ic/index.jsp?topic=%2Fcom.ibm.storage.ssic.help.doc%2Ff2c_enc_bestpractice_intro_3ekm9r.html

Keystore Considerations

It is impossible to overstate the importance of preserving your keystore data. You will not be able to decrypt your encrypted tapes if you do not have access to your keystore. Carefully read the topics below to understand the methods available for protecting your keystore data.

Attention: You cannot use both JCERACFKS and JCECCARACFKS keystore types concurrently in the Security Key Lifecycle Manager for z/OS configuration file. You must specify only one of these types in the configuration file. If the JCERACFKS and JCECCARACFKS keystore types are used concurrently, the Security Key Lifecycle Manager for z/OS server will not start.

If you plan to use a large number of keys (approximately 100,000) in either JCEKS or JCEECCAKS keystore, increase the Java heap to 512M. Otherwise, Security Key Lifecycle Manager for z/OS may not start and you may get this error message when loading the keystore:

```
Exception in thread "main" java.lang.OutOfMemoryError

at java.lang.StringBuffer, <init>(StringBuffer.java:65)

at java.lang.StringBuffer, <init>(StringBuffer.java:52)
```

The other keystores are not affected because the number of keys should never be this large.

Importance of keys and certificates

The Security Key Lifecycle Manager for z/OS and all its supported tape drives use symmetric, 256-bit AES keys to encrypt data. There are important differences in the way that TS1120, TS1130, or TS1140 and LTO Ultrium 4, or LTO Ultrium 5 tape drives handle keys and certificates. The following topics explain these differences.

Encryption Keys and the TS1120, TS1130, TS1140 Tape Drives

In addition to 256-bit AES symmetric data keys, the Security Key Lifecycle Manager for z/OS also uses public and private (*asymmetric*) key cryptography. This type of cryptography protects the symmetric data encryption keys generated and retrieved as they pass between the Security Key Lifecycle Manager for z/OS and tape drives. Public and private key cryptography also verifies the identity of the tape drives to which the Security Key Lifecycle Manager for z/OS serves keys.

When a TS1120, TS1130, or TS1140 tape drive requests a key, the Security Key Lifecycle Manager for z/OS generates a random symmetric data encryption key (DK). Public and private key cryptography wraps, or encrypts, the data encryption key (DK) using a key encryption key (KEK), the public key of an asymmetric key pair. The wrapped data key, along with key label information about what private key is required to unwrap the symmetric key. This information forms a digital envelope called an Externally Encrypted Data Key (EEDK) structure. The EEDK is stored in the tape leader area of any tape cartridge that holds data encrypted using this method. In this way, the key used to decrypt the data is stored with the data on the tape itself, protected by asymmetric, public and private key wrapping. The public key used to wrap that data key is obtained from one of two sources:

- A certificate (from a business partner, for example) stored in the Security Key Lifecycle Manager for z/OS's keystore, or

- A public key (part of an internally generated public and private key pair) stored in the Security Key Lifecycle Manager for z/OS's keystore

The DK is *only* stored on the tape, in a wrapped, protected form.

When an encrypted tape is to be read by a TS1120, TS1130, or TS1140 tape drive, the tape drive sends the EEDK to the Security Key Lifecycle Manager for z/OS. Security Key Lifecycle Manager for z/OS determines from the alias or key label which private KEK from its keystore to use, and uses the information to unwrap the EEDK and recover the DK. When the DK is recovered, it is then wrapped with a different key, which the tape drive can decrypt, and send back to the tape drive. This process enables the tape drive to decrypt the data.

The Security Key Lifecycle Manager for z/OS uses aliases, also known as key labels. The key labels identify the public and private keys that are used to wrap the EEDK when encrypting with TS1120, TS1130, or TS1140 tape drives. Specific aliases can be defined for each tape device in the device table. The Security Key Lifecycle Manager for z/OS can also be set up to apply global default aliases. For more information, see "Global default alias (key label) for TS1120, TS1130, and TS1140 tape drive writes" on page 80. If your encryption-enabled tape drives are in a tape library, you can have the library define the key labels. The key labels can then be passed directly to the tape drive. If your encryption-enabled tape drives are in a z/OS environment, you can have the key labels defined through the host. The key labels can then be passed directly to the tape drive. See *z/OS DFSMS Software Support for IBM System Storage TS1130 and TS1120 Tape Drives (3592)* for information about specifying key labels at the host.

The Security Key Lifecycle Manager for z/OS requires the definition of at least two aliases (certificates or key labels) for each encrypting tape drive. This requirement enables access to the encrypted data at another location, whether within your organization or outside it. The private key for one of these aliases must be known. If you do not want to specify two different key labels or aliases, you can define both aliases with the same value. The Security Key Lifecycle Manager for z/OS searches for two alias values in the following order:

1. From the system (for example, from DFSMS for system-managed encryption), from the library, or from the application (for example, from Tivoli Storage Manager for application-managed encryption)
2. From the drive default alias (defined in the device table)
3. From the global default alias (*drive.default.alias1* and *drive.default.alias2* in the configuration file)

The Security Key Lifecycle Manager for z/OS requires that two aliases or key labels be associated in pairs for each encrypting tape drive. Whether you use default aliases defined in the device table, global default aliases, or library-defined key labels, you must use them in pairs (#1 value and #2 value). Table 7 shows how aliases and key labels can be combined.

Table 7. Allowed alias and key label pairs. Define a minimum of one #1 and one #2.

App Key #1	Drive Default #1	Global Default #1	App Key #2	Drive Default #2	Global Default #2
X			X		
X				X	
X					X
	X		X		

Table 7. Allowed alias and key label pairs (continued). Define a minimum of one #1 and one #2.

App Key #1	Drive Default #1	Global Default #1	App Key #2	Drive Default #2	Global Default #2
	X			X	
	X				X
		X	X		
		X		X	
		X			X

Figure 7 explains how keys are processed for encrypted write operation.

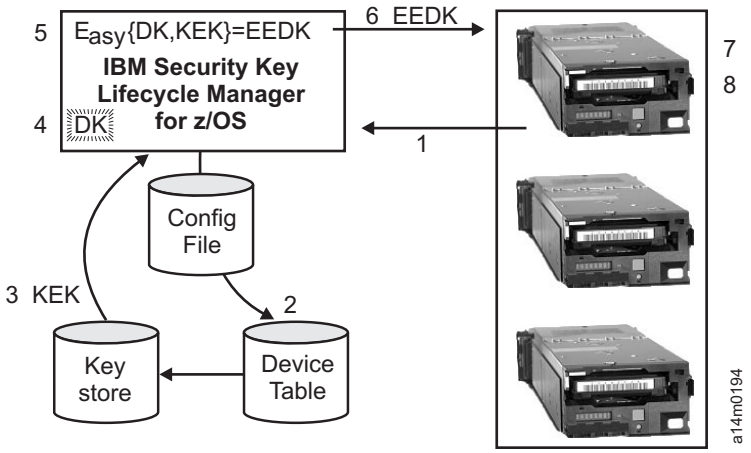

Figure 7. How the Security Key Lifecycle Manager for z/OS Responds to TS1120, TS1130, and TS1140 Tape Drive Requests for Encryption Write Operation

1. Tape drive requests key to encrypt tape
2. Security Key Lifecycle Manager for z/OS verifies tape device in Device Table
3. Security Key Lifecycle Manager for z/OS fetches keys and certificates for tape device from keystore
4. Security Key Lifecycle Manager for z/OS generates a random DK
5. Security Key Lifecycle Manager for z/OS wraps DK with public key to create an EEDK
6. Security Key Lifecycle Manager for z/OS sends the EEDK and (separately wrapped) DK to the tape drive
7. Tape drive unwraps the DK and writes the EEDK on tape leader
8. Tape drive encrypts data using DK and writes encrypted data to tape

Figure 8 on page 29 shows how keys are processed for encrypted read operation.

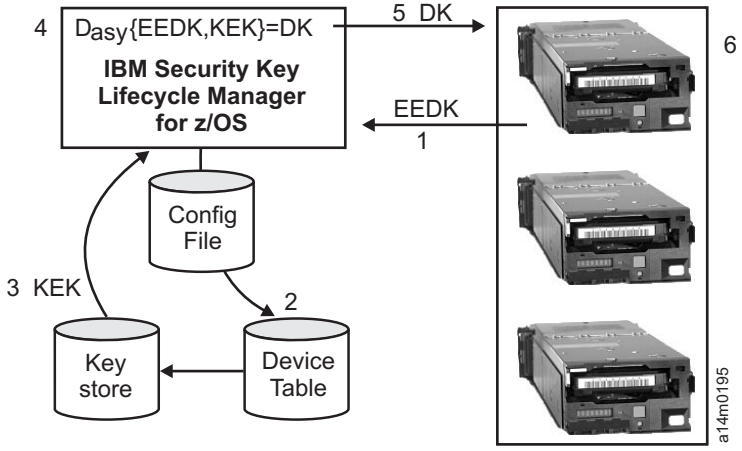

Figure 8. How the Security Key Lifecycle Manager for z/OS Responds to TS1120, TS1130. and TS1140 Tape Drive Requests for Encryption Read Operation

1. Tape drive receives read request and sends the EEDK to Security Key Lifecycle Manager for z/OS
2. Security Key Lifecycle Manager for z/OS verifies tape device in Device Table
3. Security Key Lifecycle Manager for z/OS fetches keys required to process the EEDK from keystore
4. Security Key Lifecycle Manager for z/OS unwraps EEDK using private key of KEK pair to recover DK
5. Security Key Lifecycle Manager for z/OS wraps the DK with a key the drive can decrypt and sends the wrapped DK to tape drive
6. Tape drive unwraps the DK and uses it to decrypt the data or to perform a write-append

The certificates and keys stored in the Security Key Lifecycle Manager for z/OS keystore are the point of control. The keystore allows a tape drive or library to decrypt the data on the tape. This makes the information in the keystore vital. Without the keystore, the tape cannot be read. It is important that this information is protected so that others cannot obtain the private keys from the keystore. It is also crucial that this information always is available to you so that you can read the tapes whenever necessary.

Encryption Keys and the LTO Ultrium 4 Tape Drive and LTO Ultrium 5

Only 256-bit AES symmetric data keys are used when the Security Key Lifecycle Manager for z/OS performs encryption tasks on the LTO Ultrium 4 Tape Drive and LTO Ultrium 5 Tape Drive for LTO Ultrium 4 and LTO Ultrium 5 tape cartridges.

When an LTO Ultrium 4 or LTO Ultrium 5 requests a key, the Security Key Lifecycle Manager for z/OS uses the alias specified for the tape drive. If no alias was specified for the tape drive, an alias from a key group, key alias list, or range of key aliases specified in the symmetricKeySet configuration property is used. Lacking a specific alias for the tape drive, aliases are selected from the other entities in round robin fashion to balance the use of keys evenly.

The selected alias is associated with a symmetric Data Key (DK) that was preloaded in the keystore. The Security Key Lifecycle Manager for z/OS sends this

DK to the LTO Ultrium 4 or LTO Ultrium 5 tape drive to encrypt the data. The selected alias is also converted to an entity called Data Key identifier (DKi), which is written to tape with the encrypted data. The Security Key Lifecycle Manager for z/OS can use the DKi to identify the correct DK required to decrypt the data when the LTO Ultrium 4 or LTO Ultrium 5 tape is read.

The **adddrive** and **moddrive** topics in "Command Line Interface Commands" on page 92 show how to specify an alias for a tape drive. See "Generating Keys and Aliases for Encryption on LTO Ultrium 4 and LTO Ultrium 5" on page 72. The topic explains importing keys, exporting keys, and specifying default aliases in the symmetricKeySet configuration property. "Creating and managing key groups" on page 76 shows how to define a key group and populate it with aliases from your keystore.

Figure 9 explains how keys are processed for encrypted write operation.

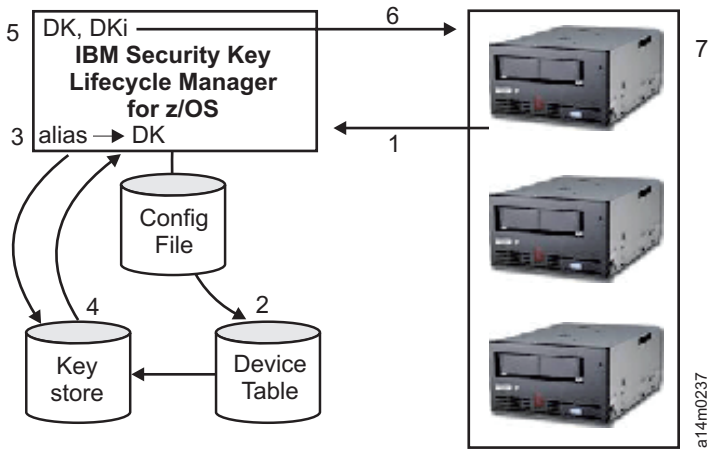

Figure 9. LTO Ultrium 4 Tape Drive and LTO Ultrium 5 Request for Encryption Write Operation

1. Tape drive requests key to encrypt tape
2. Security Key Lifecycle Manager for z/OS verifies tape device in Device Table
3. If no alias is specified in the request and no alias is specified in the device table, the Security Key Lifecycle Manager for z/OS selects an alias from the set of aliases or the key group in the keyAliasList
4. Security Key Lifecycle Manager for z/OS fetches a corresponding DK from the keystore
5. Security Key Lifecycle Manager for z/OS converts the alias to a DKi and wraps the DK with a key the drive can decrypt
6. Security Key Lifecycle Manager for z/OS sends the DK and DKi to the tape drive
7. Tape drive unwraps the DK and writes encrypted data and DKi to tape

Figure 10 on page 31 shows how keys are processed for encrypted read operation.

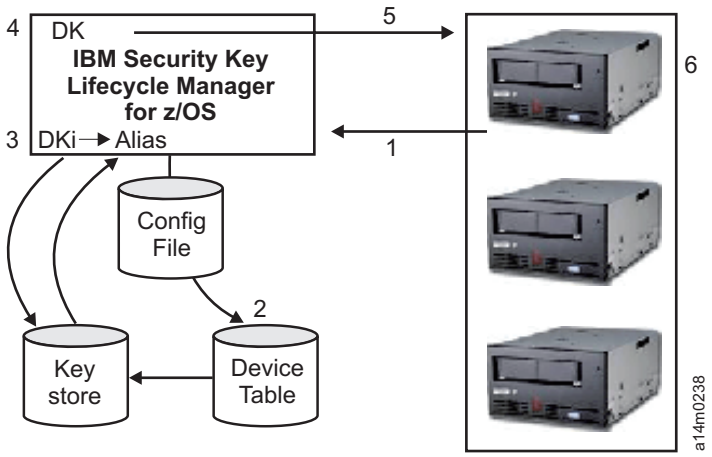

Figure 10. LTO Ultrium 4 and LTO Ultrium 5 Tape Drive Request for Encryption Read Operation

1. Tape drive receives read request and sends DKi to Security Key Lifecycle Manager for z/OS
2. Security Key Lifecycle Manager for z/OS verifies tape device in Device Table
3. Security Key Lifecycle Manager for z/OS translates DKi to alias and fetches corresponding DK from keystore
4. Security Key Lifecycle Manager for z/OS wraps the DK with a key the drive can decrypt
5. Security Key Lifecycle Manager for z/OS sends the wrapped DK to tape drive
6. Tape drive unwraps the DK and uses it to decrypt the data

Encryption Keys and the DS8000

Security Key Lifecycle Manager for z/OS also uses public and private (asymmetric) key cryptography to protect 256-bit AES symmetric unlock keys as they pass between Security Key Lifecycle Manager for z/OS and the DS8000.

When a DS8000 requests a new key, Security Key Lifecycle Manager for z/OS generates a random symmetric unlock key. Public and private key cryptography is used to wrap the unlock key using a key encryption key, which is the public key of an asymmetric key pair.

The wrapped data key, along with the key label information about the private key that is required to unwrap the symmetric key, forms a digital envelope called an Externally Encrypted Data Key (EEDK) structure that is stored on the DS8000. The key used to unlock the DS8000 is stored on the DS8000 with all the data on the DS8000, protected by asymmetric public and private key wrapping. The public key that wraps that data key is obtained from one of two sources:

- A certificate from another source such as another business partner, or a different Security Key Lifecycle Manager for z/OS which is stored in the keystore.
- A certificate with it's associated private key that was generated as part of configuration of Security Key Lifecycle Manager for z/OS and is stored in the keystore.

The certificates and keys stored in the keystore are the point of control to unlock a DS8000. Without the information in the keystore, the DS8000 cannot be unlocked.

It is important to prevent unauthorized users from obtaining the private keys from the keystore. Make sure that the keystore is always available so that you can unlock the arrays. The unlock key is stored only on the DS8000 in a wrapped, protected form.

When a DS8000 must be unlocked, the Security Key Lifecycle Manager for z/OS receives the Externally Encrypted Data Key and determines from the alias or key label which private key encryption key to use from its keystore. Security Key Lifecycle Manager for z/OS unwraps the Externally Encrypted Data Key and recovers the unlock key. After the unlock key is recovered, it is then wrapped with a different key, which the DS8000 can use to decrypt and send back to the DS8000, enabling the DS8000 to unlock disk drives.

Security Key Lifecycle Manager for z/OS uses aliases, also known as key labels, to identify the public and private keys used to wrap the unlocking key. Specific aliases can be defined for each device. You can define up to two aliases (certificates or key labels) for each DS8000 to help prevent deadlock conditions in which the Security Key Lifecycle Manager for z/OS is on the same system as the DS8000, which must unlock before the Security Key Lifecycle Manager for z/OS can start. The private key for one of these aliases must be known. If you do not want to specify two different key labels or aliases, you can define both aliases with the same value.

Figure 11 shows how keys are processed for an encrypted write operation.

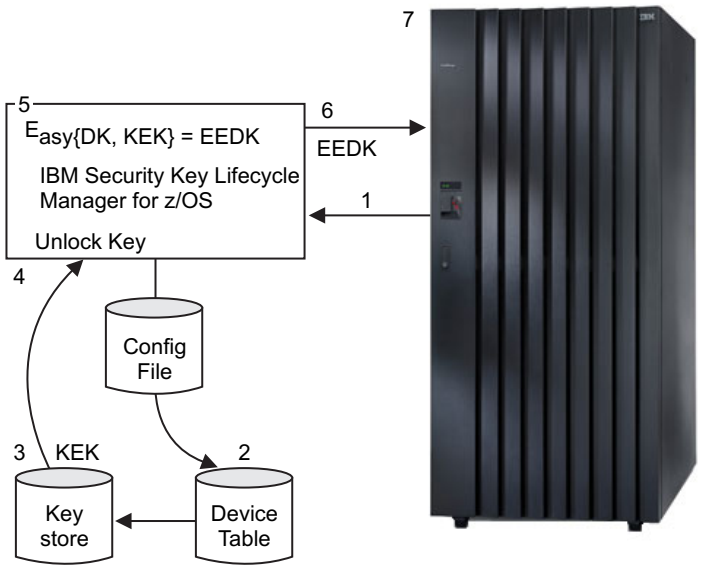

Figure 11. DS8000 Request for Encryption Write Operation

1. DS8000 requests key to unlock the drives
2. Security Key Lifecycle Manager for z/OS verifies DS8000 is in Device Table
3. Security Key Lifecycle Manager for z/OS generates a random symmetric unlock key
4. Security Key Lifecycle Manager for z/OS wraps unlock key with public key to create an EEDK
5. Security Key Lifecycle Manager for z/OS sends the EEDK and (separately wrapped) unlock key to the DS8000 drive
6. DS8000 unwraps the DK and writes the EEDK on the DS8000

7. DS8000 encrypts data using unlock key and writes encrypted data to DS8000

Figure 12 shows how keys are processed for encrypted read operation.

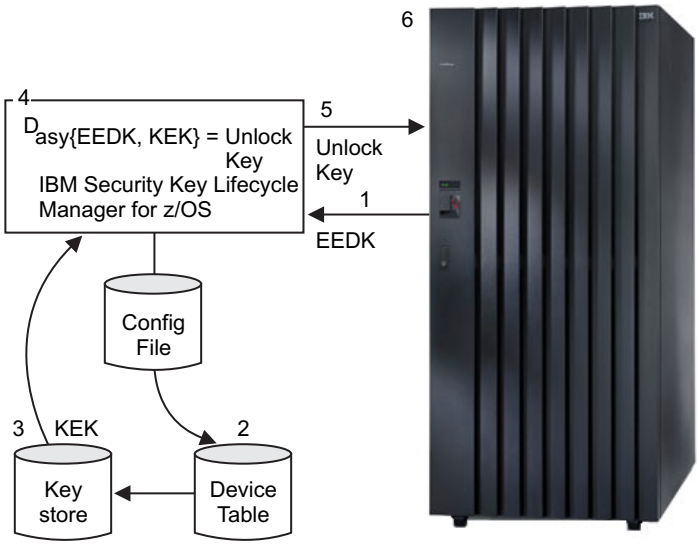

Figure 12. DS8000 Request for Encryption Read Operation

1. DS8000 receives read request and sends the EEDK to Security Key Lifecycle Manager for z/OS
2. Security Key Lifecycle Manager for z/OS verifies DS8000 is in the Device Table
3. Security Key Lifecycle Manager for z/OS fetches certificate required to process the EEDK from keystore
4. Security Key Lifecycle Manager for z/OS unwraps EEDK using private key of KEK pair to recover unlock key
5. Security Key Lifecycle Manager for z/OS wraps unlock key with a key the drive can decrypt and sends the wrapped unlock key to DS8000
6. DS8000 unwraps the unlock key and uses it to decrypt the data or to perfom a write-append

Backing up Keystore Data

Note: Due to the critical nature of the keys in your keystore, it is vital that you back up this data on a non-encrypted device. The backup is critical so that you can recover it as needed and be able to read the tapes that were encrypted using those certificates associated with that tape drive or library. Failure to back up your keystore properly can result in irrevocably losing all access to your encrypted data.

There are many ways to back up this keystore information. Each keystore type has it own unique characteristics. These are explained in more detail in "Which Keystore is Right for You" on page 36 These general guidelines apply to all:

- Keep a copy of all certificates loaded into the keystore.
- Use system backup capabilities (such as RACF) to create a backup copy of the keystore information. Be careful not to encrypt this copy using the encrypting tape drives as it would impossible to decrypt it for recovery.

- Maintain a primary and secondary Security Key Lifecycle Manager for z/OS and keystore copy (for backup and failover redundancy).
- For a JCEKS keystore, copy the keystore file and store the clear (unencrypted) copy in a secure location such as a vault. Be careful not to encrypt this copy using the encrypting tape drives as it would be impossible to decrypt it for recovery.

At a minimum, back up your keystore data whenever you change it. The Security Key Lifecycle Manager for z/OS does not modify keystore data. The only changes to the keystore are the changes that you apply, so be sure to copy the keystore as soon as you change it.

Multiple Key Lifecycle Manager for z/OS for redundancy

The Security Key Lifecycle Manager for z/OS is designed to work with supported devices. This design allows redundancy, and thus high availability. You can have multiple Security Key Lifecycle Manager for z/OS servicing the same devices. Moreover, these Security Key Lifecycle Manager for z/OSs are not required to be on the same systems as the devices. The maximum number of Security Key Lifecycle Manager for z/OS depends on your library or proxy. The only requirement is that they be available to the devices through TCP/IP connectivity.

This enables you to have two sets of Security Key Lifecycle Manager for z/OS. They are mirror images of each other. They have built-in backup of the critical information about your keystores, and as a failover in the event one Security Key Lifecycle Manager for z/OS is not available. When you configure your device (or proxy) you can point it to two sets of Security Key Lifecycle Manager for z/OS. If one Security Key Lifecycle Manager for z/OS is not available, your device (or proxy) uses the other Security Key Lifecycle Manager for z/OS.

You can also keep the two sets of Security Key Lifecycle Manager for z/OS synchronized. It is critical that you take advantage of this important function when needed. Use the feature for its inherent backup of critical data and also for its failover capability to avoid any outages in your tape operations. See "Synchronizing data between two Security Key Lifecycle Manager for z/OS servers" on page 80.

Note: Synchronization does not include keystores. They must be copied manually.

Security Key Lifecycle Manager for z/OS Server Configurations

The Security Key Lifecycle Manager for z/OS may be installed on a single-server or on multiple servers. The examples show one- and two-Security Key Lifecycle Manager for z/OS configurations but your library may allow more.

Single-Server Configuration

A single-server configuration, shown in Figure 13 on page 35, is the simplest Security Key Lifecycle Manager for z/OS configuration; because of the lack of redundancy, this configuration is not commonly used. In this configuration, all tape drives rely on a single server with no backup. If the server goes down, the keystore, configuration file, KeyGroups.xml file, and device table are unavailable, making any encrypted tape unreadable. In a single-server configuration you must ensure that back up copies of the keystore, configuration file, KeyGroups.xml file, and device table are maintained in a safe place. The files must be separate from the Security Key Lifecycle Manager for z/OS. The files must be separate so its function

can be rebuilt on a replacement server if the server copies are lost.

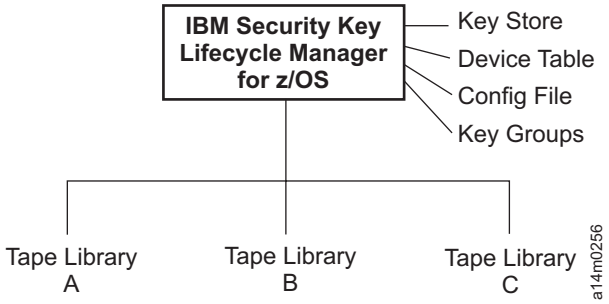

Figure 13. Single Server Configuration

Two-Server Configurations

A two-server configuration is commonly used. This configuration ensures automatic failover to the secondary Security Key Lifecycle Manager for z/OS if the primary is not accessible.

Note: When different Security Key Lifecycle Manager for z/OS servers handle requests from the same tape drives, information in the associated keystores must be identical. This requirement ensures that regardless of which server is contacted, the necessary information is available to support requests from the tape drives.

Identical configurations: For two Security Key Lifecycle Manager for z/OS servers with identical configurations, processing automatically performs a failover to the secondary server if the primary is not accessible. An example is shown in Figure 14. This configuration requires synchronized servers. Updates to the configuration file and device table of one server can be duplicated on the other automatically. You can use the **sync** command, but updates to one keystore must be copied to the other using methods specific to the keystore being used. The keystores and key groups XML file must be copied manually. See "Synchronizing data between two Security Key Lifecycle Manager for z/OS servers" on page 80 for more information.

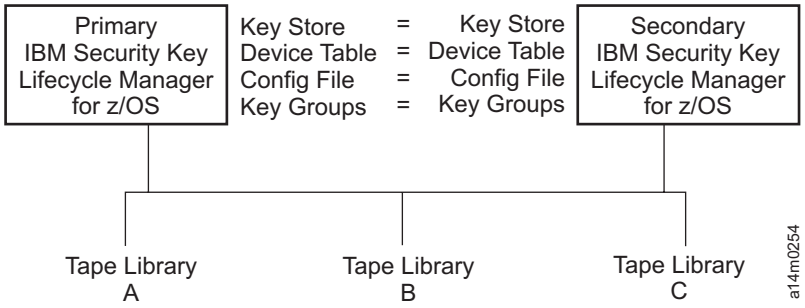

Figure 14. Two Servers with Shared Configurations

Separate configuration: Two Security Key Lifecycle Manager for z/OS servers can share a common keystore. They can also share a device table yet have two different configuration files and two different sets of key groups defined in their XML files. The only requirement is that the keys used to serve the common tape drives must be the same for each server. This requirement allows each server to have its own set of properties. An example of this type of configuration is shown in Figure 15 on page 36. Only the device table must be synchronized between servers. (See "Synchronizing data between two Security Key Lifecycle Manager for z/OS

servers" on page 80 for more information.) Be sure to specify `sync.type = drivetab` (do not specify config or all) to prevent the configuration files from being overwritten.

Note: There is no way to partially share the configuration between servers.

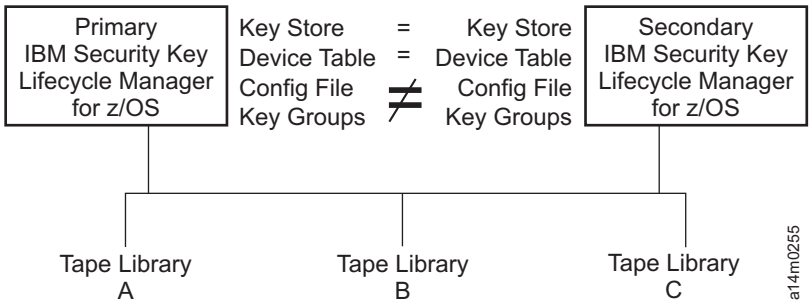

Figure 15. Two Servers with Different Configurations Accessing the Same Devices

Which Keystore is Right for You

The Security Key Lifecycle Manager for z/OS uses standard and operating system-specific Java keystore methods to store the public and private key and certificate information. This information is required to build and interpret EEDKs sent to and received from a tape drive or library. The information is also used to write and read encrypted tapes. The Security Key Lifecycle Manager for z/OS supports four keystore types. These keystore types are described to help you determine which is best for you.

Table 8. Summary of Supported Keystores

Keystore	Platform	TS1120, TS1130, TS1140, & DS8000 (store keypairs & certs)	LTO (store symmetric keys)	TS1120, TS1130, TS1140, LTO, & DS8000	Symmetric key tools available
JCEKS	z/OS	X	X	X	keytool
JCECCAKS	z/OS	X	X	X	hwkeytool
JCERACFKS	z/OS	X			N.A.
JCECCARACFKS	z/OS	X			N.A.

z/OS Keystore Options

The following keystore types are supported:

- JCEKS (UNIX System Services file based)

  A file-based keystore where Security Key Lifecycle Manager for z/OS runs. It is relatively easy to copy the contents of this keystore for backup and recovery, and to keep two Security Key Lifecycle Manager for z/OS instances synchronized for failover. JCEKS provides password-based protection of the contents of the keystore for security, and provides relatively good performance. File copy methods such as FTP can be used.
- JCECCAKS (Certificates in a file and keys can be protected by ICSF based on options chosen)

A file-based keystore supported on the z/OS platform only. This keystore can be created and managed through the JVM hwkeytool command and can create ICSF key entries. Possible hwkeytool hardwarekeytype(s) that can be defined when creating your RSA key pair are:

PKDS
The RSA private key resides in the PKDS and is protected by ICSF.

CKDS
The symmetric keys (AES, DESede) key resides in the CKDS and is protected by ICSF.

CLEAR
Key resides in a java keystore file. There is no ICSF security with using this option.

Note: In order for JCECCAKS to support symmetric keys, you must have installed SDK 50 sr6 at a minimum. If you only use JCEKS on z/OS, the minimum SDK installation is 50 sr5.
Attention: You cannot use both JCERACFKS and JCECCARACFKS keystore types concurrently in the Security Key Lifecycle Manager for z/OS configuration file. You must specify only one of these types in the configuration file. If the JCERACFKS and JCECCARACFKS keystore types are used concurrently, the Security Key Lifecycle Manager for z/OS server will not start.

- JCECCARACFKS (Certificates stored in RACF and keys protected by ICSF)

  A SAF keyring/ICSF-based keystore supported on the z/OS platform only. This keystore can be created and managed through the RACDCERT or equivalent SAF certificate management command. This keystore uses certificates generated in RACF or SAF equivalent where the key material is stored in ICSF. The JCECCARACFKS keystore uses all the security advantages of both RACF/SAF and ICSF.

  Note: This keystore type does not support symmetric keys. Therefore, to support LTO Ultrium 4 and LTO Ultrium 5 tape drives, a JCEKS, or JCECCAKS keystore must be used. If you are using only TS1120, TS1130, TS1140 tape drives or DS8000, then any of the supported keystores can be used. If you are using only LTO Ultrium 4 and LTO Ultrium 5 tape drives or using a combination of TS1120, TS1130, TS1140, DS8000, LTO Ultrium 4 and LTO Ultrium 5 tape drives with the same tape library, then a JCEKS or JCECCAKS keystore **must** be used. If you are using a combination of storage devices and you have a different tape library for each type, then you can have one Security Key Lifecycle Manager for z/OS running with an JCECCARACFKS keystore for the DS8000, TS1120, TS1130, or TS1140 tape drive or library. Another Security Key Lifecycle Manager for z/OS runs with a JCEKS or JCECCAKS keystore for the LTO Ultrium 4 and LTO Ultrium 5 tape drive/library. The two Security Key Lifecycle Manager for z/OS servers can run on the same system if they listen on different ports.

  Attention: You cannot use both JCERACFKS and JCECCARACFKS keystore types concurrently in the Security Key Lifecycle Manager for z/OS configuration file. You must specify only one of these types in the configuration file. If the JCERACFKS and JCECCARACFKS keystore types are used concurrently, the Security Key Lifecycle Manager for z/OS server will not start.

- JCERACFKS (Certificates and key material stored in RACF)

  A SAF keyring-based keystore supported on the z/OS platform only. This keystore can be created and managed through the RACDCERT or equivalent SAF management command. This keystore uses certificates generated in

RACF/SAF where the key material is stored in RACF/SAF. The JCERACFKS keystore uses all the security advantages of RACF/SAF while using software cryptography and software-based security.

Note: This keystore type does not support symmetric keys. Therefore, to support LTO Ultrium 4 and LTO Ultrium 5 tape drives, a JCEKS, or JCECCAKS keystore must be used. If you are using only TS1120, TS1130, TS1140 tape drives or DS8000, then any of the supported keystores can be used. If you are using only LTO Ultrium 4 and LTO Ultrium 5 tape drives or using a combination of TS1120, TS1130, TS1140, DS8000, LTO Ultrium 4 and LTO Ultrium 5 tape drives with the same tape library, then a JCEKS or JCECCAKS keystore **must** be used. If you are using a combination of storage devices and you have a different tape library for each type, then you can have one Security Key Lifecycle Manager for z/OS running with an JCECCARACFKS keystore for the DS8000, TS1120, TS1130, or TS1140 tape drive or library. Another Security Key Lifecycle Manager for z/OS runs with a JCEKS or JCECCAKS keystore for the LTO Ultrium 4 and LTO Ultrium 5 tape drive/library. The two Security Key Lifecycle Manager for z/OS servers can run on the same system if they listen on different ports.

Managing Keystores

When you have decided which keystore is best for your environment, you can create a keystore. If you already have a keystore, you can import keys and certificates.

Attention: You cannot use both JCECCAKS and JCECCARACFKS keystore types concurrently in the Security Key Lifecycle Manager for z/OS configuration file. You must specify only one of these types in the configuration file. If the JCECCAKS and JCECCARACFKS keystore types are used concurrently, the Security Key Lifecycle Manager for z/OS server will not start.

Keystore Passwords must not be longer than 127 Characters

The passwords for keystores in use by the Security Key Lifecycle Manager for z/OS are restricted to 127 or fewer characters. Keystore passwords 128 characters or greater in length cause the obfuscation code to fail with a NegativeArraySizeException on Security Key Lifecycle Manager for z/OS startup. This restriction is enforced as follows:

- If you are prompted for a keystore password, you must provide a password of fewer than 128 characters, otherwise, startup does not proceed.

If any keystores are already created with passwords 128 characters in length or greater, these keystore passwords can be changed using **keytool**. Keep in mind that the password cannot be changed only on the keystore itself, but must be changed for each key in the keystore. See “Changing Keystore Passwords” on page 74.

Managing Keystores on System z Platforms

For file-based (JCEKS) keystores
The standard Java tool for creating a JCEKS keystore and managing its keys and certificates is **keytool**. Visit http://www.ibm.com/developerworks/java/jdk/security/142/ for details on **keytool** usage.

For RACF keystores (keyrings) which can optionally use ICSF
The **RACDCERT** command is the interface used to create and manage keys, digital certificates, key rings, and digital certificate mappings in RACF. The RACDCERT command is documented and explained in the *z/OS Security Server*

RACF Command Language Reference. This publication can be found in the z/OS internet library at the URL: http://publibz.boulder.ibm.com/epubs/pdf/ichza460.pdf.

For ICSF-based keystores not using RACF

The Java **hwkeytool** application can create and manage the RSA keypairs and the symmetric keys (AES, or DESede) used by the devices supported by this product. See ftp://ftp.software.ibm.com/s390/java/jce4758/hwkeytool.html for details on using hwkeytool.

Disaster Recovery Site Considerations

If you plan to use a disaster recovery (DR) site, the Security Key Lifecycle Manager for z/OS provides a number of options. The options setup the DR site to read and write encrypted tapes. These options are:

- Create a duplicate Security Key Lifecycle Manager for z/OS at the DR site.

  Set up a duplicate Security Key Lifecycle Manager for z/OS at the DR site. The duplicate has the same information as your local Security Key Lifecycle Manager for z/OS (configuration file, device table, key groups XML file, and keystore). This Security Key Lifecycle Manager for z/OS would then be in place. It can take over for one of your existing production Security Key Lifecycle Manager for z/OS to read and write encrypted tapes.
- Create a backup copy of the three Security Key Lifecycle Manager for z/OS data files to be able to recover as needed.

  If you create a current copy of the four data elements needed by the Security Key Lifecycle Manager for z/OS then you can start a version at any time. You can start the software to act as a duplicate at the DR site. (Do not use the Security Key Lifecycle Manager for z/OS to encrypt the copies of these files. You must have a functioning product to decrypt files). If your DR site uses different tape drives from your primary site, the configuration file and device table must contain the correct information.
- TS1120, TS1130, or TS1140 tape drives can use the second EEDK on each tape to encrypt tapes. Such that a private key, which is unique to the DR site, is one of the entities that can read the encrypted tape. To do this setup, import the public key of the DR site. Another method is to copy their keystore. You can write an alternate certificate for the DR site. This certificate consists of using the certificate of the DR site to write your existing tapes in the same way that you provide this capability to another organization. You must store your data encryption key on your tapes. The date encryption key must be wrapped using the public and private key of your organization. In addition, the data key would also be stored on the same tapes wrapped using the public key (certificate) of the DR site. This setup allows a functioning Security Key Lifecycle Manager for z/OS at that site to use its own keystore, with its own public and private key, to read the tapes. See “Considerations for Sharing Encrypted Tapes Off-site” on page 40 for more information. If your DR site uses different tape drives from your primary site, the configuration file and device table must contain the correct information. If the information is not correct the device table is of no use at the DR site.

For DS8000, only the use of a duplicate Security Key Lifecycle Manager for z/OS at the DR site and creating a backup copy of the data files are applicable. For more information about DS8000 disaster recovery, see http://publib.boulder.ibm.com/infocenter/dsichelp/ds8000ic/index.jsp?topic=%2Fcom.ibm.storage.ssic.help.doc%2Ff2c_disasterrecover_1v0tts.html

Considerations for Sharing Encrypted Tapes Off-site

It is common practice to share tapes with other organizations for data transfer, joint development, contracting services, or other purposes. The methods for sharing encrypted tapes differ for TS1120, TS1130, TS1140 and LTO Ultrium 4, or LTO Ultrium 5 tapes.

Note: It is important to verify that any certificate received from a business partner is valid. Verify by checking the chain of trust of such a certificate back to the Certificate Authority (CA) that ultimately signed it. If you trust the CA, then you can trust that certificate. Alternately, the validity of a certificate can be verified if it was securely guarded in transit. If you do not verify the validity of a certificate in one of these ways, a "Man-in-the-Middle" attack can occur.

Sharing TS1120, TS1130, or TS1140 Tapes

The Security Key Lifecycle Manager for z/OS can store two sets of wrapped encryption keys on the IBM TotalStorage Enterprise Tape Cartridge 3592. The another organization can then read that specific tape without you providing them any shared secret information. This setup ensures that you are not compromising the security of your certificates and keys.

This is done by adding the public part of the public and private certificate and keys of the organization to your Security Key Lifecycle Manager for z/OS keystore. It is added using a second alias (or key label). When the tape is written the encryption keys are stored on the tape, protected by two sets of public and private keys, yours and the other organization's. The other organization must have an encryption-enabled TS1120, TS1130, or TS1140 Tape Drive. The organization can then use their Security Key Lifecycle Manager for z/OS. The organization can use their private key to unwrap the data key to read that specific tape.

To reiterate, your Security Key Lifecycle Manager for z/OS must have the certificate of the partner organization. The other organization must have the associated private key in the keystore used by Security Key Lifecycle Manager for z/OS of the other organization. This setup gives you the flexibility to make a specific tape readable by both your own, and another organization. If you want to take advantage of this capability you must add that certificate of the other organization which contains the public key, to your keystore. See "Encryption Keys and the TS1120, TS1130, TS1140 Tape Drives " on page 26 for more information.

Sharing LTO Ultrium 4 Tape Drive and LTO Ultrium 5 Tape Drive Tape

In order to share encrypted data on an LTO Ultrium 4 or LTO Ultrium 5 tapes, a copy of the symmetric key used to encrypt the data must be available to the other organization. This setup allows the other organization to read the tape. In order for the symmetric key to be shared, the other organization must share their public key with you. This public key is used to wrap the symmetric key when it is exported. The symmetric key is exported from the Security Key Lifecycle Manager for z/OS keystore using the keytool (see "Exporting data keys using keytool -exportseckey " on page 74). When the other organization imports the symmetric key into their Security Key Lifecycle Manager for z/OS keystore, the key is unwrapped. The symmetric key is unwrapped using their corresponding private key (see "Importing data keys using Keytool -importseckey " on page 74). This procedure ensures that the symmetric key is safe in transit since only the holder of the private key is able to unwrap the symmetric key. The other organization can then

read the data on the tape using the symmetric key used to encrypt the data in their Security Key Lifecycle Manager for z/OS.

Chapter 3. Installing the Security Key Lifecycle Manager for z/OS and Keystores

This topic gives you instructions on how to set up your z/OS environment to run the Security Key Lifecycle Manager for z/OS.

Install the Security Key Lifecycle Manager for z/OS as instructed in the Program Directory document. See, Program Directory for IBM Security Key Lifecycle Manager for z/OS.

The Security Key Lifecycle Manager for z/OS requires the IBM Java Software Developer Kit 5.0 or 6.0. See "Hardware and Software Requirements" on page 17. This topic was explained briefly in Chapter 2, "Planning your Security Key Lifecycle Manager for z/OS Environment," on page 17. There are many possible ways you can set up your Security Key Lifecycle Manager for z/OS. This section shows you how to setup keys for the four possible keystore types:

- JCEKS
- JCECCAKS
- JCERACFKS
- JCECCARACFKS

For JCECCARACFKS and JCERACFKS type keystores, it is highly encouraged that you do not use the same character alias or label names that differ only by case for example, MyKey and mykey. A search mismatch can occur when storing or retrieving information from a JCECCARACFKS and JCERACFKS keystore when using same character label or alias names differing only by case.

This topic also shows you how to run the Security Key Lifecycle Manager for z/OS in production mode.

Attention: The Security Key Lifecycle Manager for z/OS performs the function of requesting the generation of encryption keys. The product then passes those keys to the TS1120, TS1130, TS1140, LTO Ultrium 4, or LTO Ultrium 5 tape drives, and DS8000. The key material, in wrapped (encrypted) form resides in system memory during processing by the Security Key Lifecycle Manager for z/OS. The key material must be transferred without error to the appropriate tape drive so that data can be recovered (decrypted). If a corrupted key material is used to write data to a cartridge, then the data written to that cartridge cannot be recovered. There are safeguards to make sure that such data errors do not occur. If the machine hosting the Security Key Lifecycle Manager for z/OS is not using Error Correction Code (ECC) memory, the key material can become corrupted while in system memory. The corruption can then cause data loss. The chance of this occurrence is small, but for best practices use ECC memory for machines hosting critical applications.

Installing Java SDK and verifying the version

You need to install a Java SDK. See "Hardware and Software Requirements" on page 17 to understand the Java requirements for Security Key Lifecycle Manager for z/OS.

Verify the Java Version

It is important that you verify you have the correct version of Java installed in. Add the Java bin directory to your PATH, which can be done by using the USS export command as shown in the example. Replace the path with the location where your Java SDK was installed. Then issue the **java -version** command and expect to see results like shown here:

SDK 5.0:

```
export PATH=/usr/lpp/java/J5.0/bin:$PATH
java -version
java version "1.5.0"
Java(TM) 2 Runtime Environment, Standard Edition (build pmz31dev-20070426 (SR5))
IBM J9 VM (build 2.3, J2RE 1.5.0 IBM J9 2.3 z/OS s390-31 j9vmmz3123-20070426 (JIT enabled)
J9VM - 20070420_12448_bHdSMr
JIT - 20070419_1806_r8
GC - 200704_19)
JCL - 20070425
```

SDK 6.0

```
export PATH=/usr/lpp/java/J6.0/bin:$PATH
java -version
java version "1.6.0"
Java(TM) SE Runtime Environment (build pmz3160sr6-20091029_01(SR6))
IBM J9 VM (build 2.4, JRE 1.6.0 IBM J9 2.4 z/OS s390-31 jvmmz3160sr6-20091028_45330 (JIT enabled, AOT enabled)
J9VM - 20091028_045330
JIT  - r9_20090902_1330ifx1
GC   - 20090817_AA)
JCL  - 20090924_01
```

Another aspect to consider is which platforms and SDK levels your partners will use to read the tapes written from your z/OS system. This specification ensures that the cryptographic capabilities of the reader are compatible with the SDK level that you have chosen for your z/OS Security Key Lifecycle Manager for z/OS deployment.

Copying the unrestricted policy files

You must replace the US_export_policy.jar and local_policy.jar files in the $JAVA_HOME/lib/security directory with an unrestricted version of these files. These unrestricted policy files are required by the Security Key Lifecycle Manager for z/OS in order to serve AES keys.

On z/OS, copy the unrestricted policy files that are shipped in the z/OS Java SDK build under the jce demo directory. Copy them to the **lib/security** directory as shown in this example:

```
cp /usr/lpp/java/J5.0/demo/jce/policy-files/unrestricted/*
/usr/lpp/java/J5.0/lib/security
```

Alternatively, the unrestricted policy files can be downloaded from the following website:https://www14.software.ibm.com/webapp/iwm/web/preLogin.do?source=jcesdk.

Be sure to select the unrestricted JCE policy files for the SDK you are currently using. See https://www14.software.ibm.com/webapp/iwm/web/preLogin.do?source=jcesdk for the correct unrestricted policy files.

Add the Java Hardware Provider (Only if Using ICSF)

If you are not sure what keystore type you want to use, read "Which Keystore is Right for You" on page 36. If you decided to use a keystore of type JCECCAKS or JCECCARACFKS to use the security advantages of ICSF, you must add the Java hardware provider.

Attention: You cannot use both JCERACFKS and JCECCARACFKS keystore types concurrently in the Security Key Lifecycle Manager for z/OS configuration file. You must specify only one of these types in the configuration file. If the JCERACFKS and JCECCARACFKS keystore types are used concurrently, the Security Key Lifecycle Manager for z/OS server will not start.

Note: For test purposes, you can use the JCEKS software keystore. This keystore does not use ICSF and thus does not require you to add the Java hardware provider at this time.

To add the Java hardware provider, you must edit the **$JAVA_HOME/lib/security/java.security** file. You can then add the hardware provider so that it is the second provider in the list as shown in the examples shown. Be sure to change the security.provider.# so that the providers are listed in order from 1, 2, 3.

For SDK 5.0 and higher, add the IBMJCECCA provider for best practices:

```
#
# List of providers and their preference orders (see above):
#
security.provider.1=com.ibm.jsse2.IBMJSSEProvider2
security.provider.2=com.ibm.crypto.hdwrCCA.provider.IBMJCECCA
security.provider.3=com.ibm.crypto.provider.IBMJCE
security.provider.4=com.ibm.security.jgss.IBMJGSSProvider
security.provider.5=com.ibm.security.cert.IBMCertPath
security.provider.6=com.ibm.security.sasl.IBMSASL
```

For more detailed information about the Java hardware provider, see http://www.ibm.com/servers/eserver/zseries/software/java/jcecca14.html.

Setting up a user ID to run the Security Key Lifecycle Manager for z/OS

The Security Key Lifecycle Manager for z/OS requires a z/OS user ID that identifies the Security Key Lifecycle Manager for z/OS process to z/OS. For production deployments, launch Security Key Lifecycle Manager for z/OS using the JZOS launcher, see "Setting up and running Security Key Lifecycle Manager for z/OS in Production Mode" on page 67. In addition, the Security Key Lifecycle Manager for z/OS must be able to retrieve the private key of your X.509 Digital Certificate. The private key must be retrieved when servicing Tape Write and Read requests. For RACF type keystores the user ID under which the Security Key Lifecycle Manager for z/OS runs must be the owner of the certificate. See *z/OS Security Server Security Administrator's Guide* for an explanation of the rules that govern access to private keys and certificates.

In all the following examples, the user ID of ISKLMSRV is used.

This user ID must have an OMVS segment with a UID and GID defined. The UID need not be zero and can be any value. The home directory in this user ID's OMVS segment is where the Security Key Lifecycle Manager for z/OS is started. The user ID must also run the standard shell at login (/bin/sh), and be connected to a

default group that has a GID. You can allow RACF to automatically assign the UID or explicitly define the UID. The ISKLMSRV user ID is a protected user. It cannot be logged on to.

Note: In OMVS, the configuration file permission is set so that only the owner can read or write the configuration file. If you log on and you are not the owner of the configuration file, you do not have permission to write to the configuration file. You might encounter an error similar to this: `- java.io.FileNotFoundException: /u/isklmsrv/JA0/ISKLMConfig.properties.zos.JCECCARACFKS (EDC5111I Permission denied.)` You might encounter this error when stopping the server, running the refresh operation, or changing passwords. For best practices, log on using the user ID with owner permissions.

The use of italics indicates fields that you can customize in this example.

```
AU ISKLMSRV DFLTGRP(SYS1) OMVS(AUTOUID HOME(/u/ISKLMSRV)PROGRAM(/bin/sh))
  NOPASSWORD NOOIDCARD
```

Obtaining Digital Certificates

Before starting your Security Key Lifecycle Manager for z/OS, you must have at least one X.509 digital certificate (contains a public and private key pair). The digital certificate is to protect the data encryption key that the Security Key Lifecycle Manager for z/OS creates when encrypting data to tape. The use of certificates, their public key, and the corresponding private key is explained in "Importance of keys and certificates" on page 26. The Security Key Lifecycle Manager for z/OS allows for two digital certificate aliases to be defined per write request. One of the two aliases/labels specified must have a private key in the keystore of Security Key Lifecycle Manager for z/OS when the tape is created. This guarantees that the creator of the tape can read the tape. The other label/alias can be a public key from a business partner which they can decrypt with their private key. In order to read an encrypted tape, the correct private key is needed.

There are two methods of setting up digital certificates:

- Creating your own public and private key pair and corresponding certificate. Those keys and certificates are used to write/encrypt to tape so that you can read/decrypt the tape at a later date.
- Obtaining the public key of a business partner and corresponding certificate. Those keys and certificates are used to write/encrypt tapes that can be read/decrypted by your business partner.

Note: The Security Key Lifecycle Manager for z/OS does not read certificates with NO-TRUST status. To verify the status with RACF, issue a RACDCERT LIST command to display the certificate. This pertains to ACF2 and other security products as well. This is only applicable to JCERACFKS and JCECCARACFKS.

Creating Your Own Public and Private Key Pair and Corresponding Certificate

There are several ways to setup your own public and private key pair for use by the Security Key Lifecycle Manager for z/OS:

Using Certificates You Already Have

You might already have certificates and their associated public and private keys that are suitable for using with Security Key Lifecycle Manager for z/OS. These certificates can be used as long as they are one of the following:

- Existing certificate and corresponding public and private keys that can be exported and then imported into Java JCEKS or JCECCAKS keystore. A second copy of the key would then exist in a keystore file or PKDS.
- Existing RACF keys that are connected to RACF key rings so they can be accessed by the Security Key Lifecycle Manager for z/OS.

Note: Keys with no corresponding X.509 certificate cannot be used with the Security Key Lifecycle Manager for z/OS. For example, an ICSF public and private key that was created using ICSF tools does not have a corresponding certificate.

Generating a new public and private key pair and corresponding certificate

You can generate a new public and private key pair and corresponding certificate to be used exclusively for processing and protecting your data on tape. It can be a self-signed certificate, a certificate signed using an internal z/OS Certificate Authority. It can also be a certificate signed by a third-party Certificate Authority such as VeriSign.

There are several tools you can use to create the public and private key pair and certificate. The examples in this publication illustrate how to create keys using:

- Java keytool (using software encryption)
- hwkeytool (using z/OS cryptography provided by ICSF)
- RACF's RACDCERT command (usings RACF and, optionally, ICSF)

Note: You can also use an equivalent z/OS security product other than the IBM z/OS Security Server RACF product. If you do, consult the publications associated with that product. Find the functionally equivalent operations and steps with respect to the RACF RACDCERT commands shown in the following examples.

Obtaining a public key and corresponding certificate from a business partner

You can exchange encrypted tapes with a business partner. To write encrypted tapes to be sent to a business partner, you must import a public key/certificate from your business partner. They can then read the encrypted tape with their corresponding private key. The reverse is also true. For a business partner to create encrypted tapes that you can read, you must export a public key/certificate from one of your public and private key pairs. You can then read the encrypted tape with your private key. The following examples illustrate how you would export and import a public key/certificate using the various tools (Java keytool, java hwkeytool and RACF's RACDCERT).

Examples of How to Set Up Digital Certificates

The examples show how you can use the Java tools (automatically available to you in the Java installation) and the RACF RACDCERT command. The examples show you how to set up digital certificates for use by Security Key Lifecycle Manager for z/OS. They are organized by the keystore type you select to start the Security Key Lifecycle Manager for z/OS. For more information see “Which Keystore is Right for You” on page 36).

- “Example 1: Using the Java Keytool and JCEKS on z/OS” on page 48 (software keys)
- “Example 2: Using the JCECCAKS Keystore with the Java Hwkeytool on z/OS” on page 50 (ICSF keys not using RACF)

- "Example 3: Using the JCERACFKS or JCECCARACFKS Keystore on z/OS" on page 53 (RACF keys which are not in ICSF)

Example 1: Using the Java Keytool and JCEKS on z/OS

The JCEKS keystore is a file-based keystore. If you use this keystore it is relatively easy to copy the contents of this keystore. Copy the keystore for back up and recovery, and keep two Security Key Lifecycle Manager for z/OS instances synchronized for failover. JCEKS provides password-based protection of the contents of the keystore and provides relatively good performance. The keystore is protected by the z/OS file system and z/OS security product. The Java keytool provides management of the JCEKS-based keystore and its contents. You can manage the private keys and their associated X.509 certificates, and the certificate chains that authenticate the authenticity of a certificate. For more information about the Java keytool, see "Managing Keystores on System z Platforms" on page 38.

Verify Java is in Your Path

To use the keytool command, make sure that Java is in your path. See "Verify the Java Version" on page 44.

Generate a Public and Private Key and Corresponding Certificate for Your Security Key Lifecycle Manager for z/OS

This example generates a public and private key pair and associated self-signed certificate, which is stored in a keystore. The resulting keystore can be used by your Security Key Lifecycle Manager for z/OS to Write and Read encrypted tapes. You can use this key to define a default alias in your **ISKLMConfig.properties.zos** file (drive.default.alias1 = CERT1).

Alternately, a certificate request can be generated and submitted to a third-party certificate authority (or internal z/OS certificate authority). The fulfilled certificate and the certificate authority certificate can be imported back into a JCEKS keystore for use by the Security Key Lifecycle Manager for z/OS.

For information about using the Java keytool to generate a certificate request and import the results back onto your JCEKS keystore, see this website publication: https://www.ibm.com/developerworks/java/jdk/security/50/secguides/keytoolDocs/KeyToolUserGuide-150.html.

For information about how to use SSL to create an internal z/OS Certificate Authority see, *z/OS Cryptographic Services System Secure Sockets Layer Programming* SC24-5901.

Generate an RSA private and public key pair and associated certificate. The keypass and storepass values must be the same. This is because the Security Key Lifecycle Manager for z/OS allows you to specify only one password to the keystore (in the ISKLMConfig.properties.zos "config.keystore.password ="). It does not provide a way to send a specific password in with each keylabel.

In this example:

- Key alias is CERT1.
- Distinguished name of the subject is myCo.
- Keystore file name is ISKLMKeystore.
- Password used to protect this keystore is 'somesecretphrase'.

- Expiration of the certificate is 999 days.
- Key size generated is 2048 bits in length.

Type this command:

```
keytool -genkey -alias CERT1 -dname "CN=myCo"
-keystore ISKLMKeystore -provider IBMJCE -keyalg RSA -keysize 2048
-keypass "somesecretphrase" -storepass "somesecretphrase"
-storetype JCEKS  -validity 999
```

To list the contents of the keystore where the certificate was created, type this command:

```
keytool -list -keystore ISKLMKeystore -storetype JCEKS
-storepass "somesecretphrase"
```

Exchange a Public Key or Certificate with a Business Partner

For example, you can export the CERT1 self-signed certificate and public key to a business partner, so that the business partner can write encrypted tapes that can be read by your Security Key Lifecycle Manager for z/OS. In this example, you send the public key and corresponding certificate (alias/label CERT1) that was created in the previous example for use by your Security Key Lifecycle Manager for z/OS. You send only the public key (not the private key) so that there is no security compromise.

Export the self-signed certificate & public key “CERT1” to a file called **ExportedPublicKey.cer**.

```
keytool -export -file ExportedPublicKey.cer -keystore ISKLMKeystore
-alias CERT1 -storepass "somesecretphrase" -storetype JCEKS
-provider IBMJCE
```

Print the newly created certificate file using the keytool printcert utility.

```
keytool -printcert -file ExportedPublicKey.cer
```

Now you can send the **ExportedPublicKey.cer** file to your business partner. Tell them the alias “CERT1” if they are not able to use an encoding mechanism of Public Key Hash, explained in more detail in this topic.

Import the certificate and public key from your business partner.

Note: The *alias* you specify on import must match the *alias* that was used by the business partner (in this example, CERT1). This case is true if you plan to specify an encoding mechanism of Label “L” when encrypting tapes. Optionally, you can specify an encoding mechanism of Public Key Hash “H” that will use a Hash value rather than the KeyLabel to identify the key. While Hash gives slightly less performance, it allows you to import a certificate/public key from a business partner. It does so without knowing the alias/KeyLabel the business partner used to create/export the key. It also gives you the freedom to specify the label you want to use to identify the public key of your business partner. Therefore using Public Key Hash would be the preferred method. An example of how the z/OS encoding mechanism is provided in this section.

Import the certificate and public key from the business partner contained in ExportedPublicKey.cer to the ISKLMKeystore with an alias KEYS1. In this example the imported alias of KEYS1 does not match the original business partner's alias of CERT1. This only works if you plan to specify an encoding mechanism of Public

Key Hash "H" for this key. This is shown in the example following this import example. Otherwise the alias on the import command must be provided to you by your business partner.

```
keytool -import  -file ExportedPublicKey.cer -keystore ISKLMKeystore
-alias KEYS1 -storepass "somesecretphrase"
-storetype JCEKS -provider IBMJCE -keypass "somesecretphrase"
```

List the contents of the keystore the certificate was imported to:

```
keytool -list -keystore ISKLMKeystore -storetype JCEKS -storepass "somesecretphrase"
```

The encoding key mechanism can be specified in the data class or JCL. The example shows how it would be specified in the JCL. As shown in this example, it is best that you specify an encoding mechanism of Label "L" when defining your own Key. However, you should specify an encoding mechanism of Public Key Hash "H" when defining a business partner key.

```
//C02STRW1 JOB CONSOLE,
// MSGCLASS=H,MSGLEVEL=(1,1),CLASS=B,
// TIME=1440,REGION=2M
/*JOBPARM SYSAFF=*
//*
//* ENC KEY MASTER JOB
//*
//CREATE1 EXEC PGM=IEBDG
//SYSPRINT DD SYSOUT=*
//SEQ001 DD DSN=TAPE.C02M5CX2.PC5.NOPOOL.C02STRS1.MASTER,
// KEYLABL1='CERT1',
// KEYENCD1=L,
// KEYLABL2='KEYS1',
// KEYENCD2=H,
// LABEL=(1,SL),UNIT=C02M5CX2,DISP=(,CATLG),
// DCB=(DSORG=PS,RECFM=FB,LRECL=2048,BLKSIZE=6144)
//SYSIN DD *
DSD OUTPUT=(SEQ001)
FD NAME=A,STARTLOC=1,LENGTH=10,FORMAT=ZD,INDEX=1
FD NAME=B,STARTLOC=11,LENGTH=13,PICTURE=13,'PRIMER RECORD'
CREATE QUANTITY=25,FILL='Z',NAME=(A,B)
END
/*
```

For more information about the Hash encoding mechanism, see *z/OS DFSMS Software Support for IBM System Storage TS1130 and TS1120 Tape Drives (3592)*, SC26-7514.

Example 2: Using the JCECCAKS Keystore with the Java Hwkeytool on z/OS

If you use the JCECCAKS keystore, you can take advantage of ICSF Security, which is supported only on the z/OS platform. This keystore is a file-based keystore. The certificate and public key are stored in a file. The private key can be stored in ICSF depending on the option specified when the key is created or imported. For information about ICSF back up and recovery and Master Key management see, *z/OS Cryptographic Services Integrated Cryptographic Service Facility System Programmer's Guide*, and *z/OS Cryptographic Services Integrated Cryptographic Service Facility Administrator's Guide*.

Attention: You cannot use both JCERACFKS and JCECCARACFKS keystore types concurrently in the Security Key Lifecycle Manager for z/OS configuration file. You must specify only one of these types in the configuration file. If the JCERACFKS and JCECCARACFKS keystore types are used concurrently, the Security Key Lifecycle Manager for z/OS server will not start.

The Java hwkeytool provides management of the JCECCAKS-based keystore. For more information about the Java hwkeytool, see “Managing Keystores on System z Platforms” on page 38.

Verify Java is in Your Path

To use the keytool command, make sure that Java is in your path. See “Verify the Java Version” on page 44.

ICSF Must be Started

The ICSF address space must be started for the key generation to be successful.

Generate a Public and Private Key and Corresponding Certificate for Your Security Key Lifecycle Manager for z/OS

This example generates a public and private key pair and associated self-signed certificate where the private key is stored in the ICSF PKDS. The resulting keystore can be used by your Security Key Lifecycle Manager for z/OS to Write and Read encrypted tapes. You can use this key to define a default alias in your config file (that is, drive.default.alias1 = CERT2).

Alternately, a certificate request can be generated and submitted to a third-party certificate authority (or internal z/OS certificate authority). The fulfilled certificate and the certificate authority certificate can be imported back into a JCECCAKS keystore for use by the Security Key Lifecycle Manager for z/OS.

For information about using the Java keytool, see http://www-128.ibm.com/developerworks/java/jdk/security/142/secguides/keytoolDocs/KeyToolUserGuide-142.html#certreqCmd.

For information about how to use SSL to create an internal z/OS Certificate Authority see *z/OS Cryptographic Services System Secure Sockets Layer Programming* SC24-5901.

Generate an RSA private/public key pair and associated certificate where the private key is stored in the ICSF PKDS by specifying the –hardwaretype PKDS option. The default –hardwaretype (when no option is specified) is CLEAR. It does not store the private key in the ICSF PKDS but rather in the Java keystore file where the public key and certificate are being stored (in this example, ISKLMKeystoreCCA). The keypass and storepass values must be the same. This is because the Security Key Lifecycle Manager for z/OS allows you to specify only one password to the keystore (in the ISKLMConfig.properties.zos "config.keystore.password ="). It does not provide a way to send a specific password in with each keylabel. In this example, the key alias is CERT2, the distinguished name of the subject is myCo, the keystore file name is ISKLMKeystoreCCA. The password used to protect this keystore is 'somesecretphrase' and the expiration of the certificate is 999 days. The key size generated is 2048 bits in length.

```
hwkeytool -genkey  -alias CERT2  -dname "CN=myCo"
-keystore ISKLMKeystoreCCA -provider IBMJCECCA  -keyalg RSA
-keysize 2048 -storetype JCECCAKS -keypass "somesecretphrase"
-storepass "somesecretphrase" -hardwaretype PKDS
```

List the contents of the keystore where the certificate was created.

```
hwkeytool -list  -keystore ISKLMKeystoreCCA -storepass "somesecretphrase"
-storetype JCECCAKS -provider IBMJCECCA
```

Note: Specify the Correct provider and storetype for your Java SDK.

You may modify the **hwkeytool** commands in the example. For best practices, use –provider IBMJCECCA and –storetype JCECCAKS when using Java SDK 5.0 and higher.

Exchange a Public Key or Certificate with a Business Partner

The example shows how to export a certificate and public key to a business partner. It also shows how the business partner can import it so that the business partner can write encrypted tapes that can be read by your Security Key Lifecycle Manager for z/OS. In this example the public key and corresponding certificate (alias/label CERT2) that was created in the previous example are sent for use by your Security Key Lifecycle Manager for z/OS. Only send the public key (not the private key) so there is no security compromise.

Export the self-signed certificate & public key “CERT2” to a file called ExportedPublicKey.cer.

```
hwkeytool -export -file ExportedPublicKey.cer -keystore ISKLMKeystoreCCA
-alias CERT2 -storepass "somesecretphrase"
-storetype JCECCAKS -provider IBMJCECCA
```

Print the newly created certificate file using the hwkeytool printcert utility.

```
hwkeytool -printcert -file ExportedPublicKey.cer
```

Now you can send the **ExportedPublicKey.cer** file to your business partner. You might also tell them the alias “CERT2” if they are not able to use an encoding mechanism of Public Key Hash. This concept is explained in more detail in this topic.

Import the certificate and public key from your business partner:

Note: The *alias* you specify on import must match the *alias* that was used by the business partner (in this example, CERT2). This is true if you plan to specify an encoding mechanism of Label “L” when encrypting tapes. Optionally, you can specify an encoding mechanism of Public Key Hash “H” that will use a Hash value rather than the KeyLabel to identify the key. While Hash gives slightly less performance, it allows you to import a certificate/public key from a business partner. It does so without knowing the alias/KeyLabel the business partner used to create/export the key. It also gives you the freedom specify the label you want to use to identify the public key of your business partner. Therefore using Public Key Hash would be the preferred method. An example of how the z/OS encoding mechanism is specified is provided in this section.

Import the certificate and public key from the business partner contained in ExportedPublicKey.cer to the ISKLMKeystoreCCA with an alias KEYS2. In this example the imported alias of KEYS2 does not match the original business partner's alias of CERT2. This only works if you plan to specify an encoding mechanism of Public Key Hash “H” for this key. This is shown in the example following this import example. Otherwise the alias on the import command must be provided to you by your business partner.

```
hwkeytool -import -file ExportedPublicKey.cer -keystore ISKLMKeystoreCCA
-alias KEYS2 -storepass "somesecretphrase"
-storetype JCECCAKS -provider IBMJCECCA -keypass "somesecretphrase"
```

List the contents of the keystore the certificate was imported to:

```
hwkeytool -list -keystore ISKLMKeystoreCCA -storetype JCECCAKS
-storepass "somesecretphrase"
```

The encoding key mechanism can be specified in the data class or JCL. The example shows how it would be specified in the JCL. As is shown in this example, it is best that you specify an encoding mechanism of Label "L" when defining your own Key. However, you should specify an encoding mechanism of Public Key Hash "H" when defining a business partner key.

```
//C02STRW1 JOB CONSOLE,
// MSGCLASS=H,MSGLEVEL=(1,1),CLASS=B,
// TIME=1440,REGION=2M
/*JOBPARM SYSAFF=*
//*
//* ENC KEY MASTER JOB
//*
//CREATE1 EXEC PGM=IEBDG
//SYSPRINT DD SYSOUT=*
//SEQ001 DD DSN=TAPE.C02M5CX2.PC5.NOPOOL.C02STRS1.MASTER,
// KEYLABL1='CERT2',
// KEYENCD1=L,
// KEYLABL2='KEYS2',
// KEYENCD2=H,
// LABEL=(1,SL),UNIT=C02M5CX2,DISP=(,CATLG),
// DCB=(DSORG=PS,RECFM=FB,LRECL=2048,BLKSIZE=6144)
//SYSIN DD *
DSD OUTPUT=(SEQ001)
FD NAME=A,STARTLOC=1,LENGTH=10,FORMAT=ZD,INDEX=1
FD NAME=B,STARTLOC=11,LENGTH=13,PICTURE=13,'PRIMER RECORD'
CREATE QUANTITY=25,FILL='Z',NAME=(A,B)
END
/*
```

For more information about the Hash encoding mechanism, see *z/OS DFSMS Software Support for IBM System Storage TS1130 and TS1120 Tape Drives (3592)*, SC26-7514 publication.

Example 3: Using the JCERACFKS or JCECCARACFKS Keystore on z/OS

If you use the JCERACFKS or JCECCARACFKS keystore you can take advantage of RACF or both RACF and ICSF Security. They are supported only on the z/OS platform. This is a keyring-based keystore. The certificate and public and private key pair are stored in RACF. The public and private key pair can optionally be stored in ICSF depending on the option specified when the key is created or imported.

Attention: You cannot use both JCERACFKS and JCECCARACFKS keystore types concurrently in the Security Key Lifecycle Manager for z/OS configuration file. You must specify only one of these types in the configuration file. If the JCERACFKS and JCECCARACFKS keystore types are used concurrently, the Security Key Lifecycle Manager for z/OS server will not start.

The JCERACFKS uses SAF and RACF services to protect key material and certificates. The JCECCARACFKS keystore uses SAF and RACF services with the addition of ICSF to protect certificates and key material. For SAF/RACF-stored key rings, the RACF RACDCERT command is the interface used to manage the

keyring. The RACDCERT command is documented in the *z/OS Security Server RACF Command Language Reference* publication for more information, seehttp://publibz.boulder.ibm.com/epubs/pdf/ichza460.pdf.

For information about ICSF back up and recovery and Master Key management, see:

- *z/OS Cryptographic Services Integrated Cryptographic Service Facility System Programmer's Guide*
- *z/OS Cryptographic Services Integrated Cryptographic Service Facility Administrator's Guide*

The examples illustrate several ways to generate a public and private key pair and associated certificate where the public and private key pair is stored in the ICSF PKDS. The resulting certificate is then connected to a keyring which can be used by the Security Key Lifecycle Manager for z/OS to Write and Read encrypted tapes. This example also shows how to export the certificate and public key and import them so that they can be used by a business partner. The business partner can use it to write encrypted tapes that can be read by your Security Key Lifecycle Manager for z/OS.

Define a Keyring

You must define a keyring for the Security Key Lifecycle Manager for z/OS Server user ID on z/OS when using the JCERACFKS or JCECCARACFKS keystore types. Use the zOS RACF **RACDCERT** command as shown in the example. **RACDCERT** creates the keyring with the name of ISKLMRing for the user ID ISKLMSRV.

It is assumed that this command is being issued by an administrator who has authority to issue the **RACDCERT** command. See *z/OS Security Server RACF Command Language Reference* for additional information and a detailed explanation of the **RACDCERT** command and parameters. If you are using another z/OS security product, you must perform this task using the tooling that is appropriate for that security product.

```
RACDCERT ID(ISKLMSRV) ADDRING(ISKLMRing)
```

Ensure that the Security Key Lifecycle Manager for z/OS server is authorized to read from its keyring, and is authorized to use the key ICSF key label. Verify that the required RACF FACILITY class profiles are defined. If it is not defined issue the **RDEFINE** commands as shown to define these profiles which protect the use of keyring functions:

```
RDEFINE FACILITY IRR.DIGTCERT.LIST UACC(NONE)
RDEFINE FACILITY IRR.DIGTCERT.LISTRING UACC(NONE)
PERMIT IRR.DIGTCERT.LIST CLASS(FACILITY) ID(ISKLMSRV) ACC(READ)
PERMIT IRR.DIGTCERT.LISTRING CLASS(FACILITY) ID(ISKLMSRV) ACC(READ)
```

Generate a Certificate/RSA Key Pair

This example generates an RSA key pair and self-signed certificate. It then stores the RSA keys in the ICSF PKDS. The ICSF address space must be started for the key generation to be successful. The PCICC parameter of the RACDCERT GENCERT command indicates to RACF that the cryptographic hardware is to be used in the creation of the key pair. The SIZE indicates the modulus to be used in the creation of key pairs. The key label for the PKDS that is specified on the PCICC parameter must follow the z/OS data set naming conventions.

Note: The use of ICSF is optional. If you choose not to use ICSF, the key that you generate is saved in the RACF database and limited in strength to 1024 bits. This setting can be used in a testing environment.

You can use the self-signed certificate as is. You can then have it signed by a self-signed certificate authority certificate within RACF. You can optionally choose to export a certificate request and submit to a third-party certificate authority if your installation uses a third-party certificate authority. All three cases are illustrated in the examples in this section.

Use the RACDCERT **GENCERT** command to generate the RSA public and private key pair and create a self-signed certificate. In this example, the Security Key Lifecycle Manager for z/OS instance on z/OS will be executing with a z/OS user ID of ISKLMSRV. The subject distinguished name in the certificate that identifies this Security Key Lifecycle Manager for z/OS instance has a common name of ITOperations, for the company MyCo, in the United States. The certificate has a label associated with it of ISKLMServer to easily identity the certificate.

The RACDCERT **GENCERT** command has two possible keywords, **PCICC** (used in the following examples) and **ICSF**. Both stores the private key in the PKDS of ICSF with the following differences:

PCICC keyword
ICSF subsystem must be operational and configured for PKA operations.

A PCI-class cryptographic coprocessor must be operational.

The private key generated with RSA algorithm and stored as an ICSF RSA Chinese Remainder Theorem (CRT) key token in the PKDS.

ICSF keyword
ICSF subsystem must be operational and configured for PKA operations.

The private key is generated with RSA algorithm and stored as an ICSF RSA Modulus-Exponent (ME) key token in the PKDS.

For additional information see the RACF and ICSF documentation mentioned earlier in this document.

1. **Generating a self-signed certificate**
   a. Generate an RSA key pair and certificate for the Security Key Lifecycle Manager for z/OS server instance on z/OS. The following RACDCERT command illustrates that the certificate generated uses ICSF for private key storage. The key size is 2048 bits and the private key is saved in the ICSF PKDS in an encrypted form.

```
RACDCERT GENCERT SUBJECTSDN(CN('ITOperations')
O('MyCo') C('US')) WITHLABEL('ISKLMServer')
PCICC(ITOPS.ISKLM.CERT) SIZE(2048)
```

   If you are not using ICSF omit the PCICC keyword and change the keysize to 1024.

   Note: This certificate can be used as is as a self-signed certificate or submitted to a third-party certificate provider for signing. See item 3 on page 57 for information about submitting to a third-party certificate provider.
   b. You can send this certificate to other business partners or sites within your enterprise. This is so that the certificate that identifies the Security Key Lifecycle Manager for z/OS instance on z/OS is known to your partners.

By using this self-signed certificate, your business partners or remote sites agree to trust this certificate. This certificate can be imported into the keystore that is being used by the Security Key Lifecycle Manager for z/OS at your location of your partner. To send this certificate, you must export it to a dataset

```
RACDCERT EXPORT (LABEL('ISKLMServer'))
DSN('hlq.PUBKEY.S2048.ITOPS') FORMAT(CERTDER)
```

See "Business Partner and Remote z/OS Systems " on page 58 for information.

c. You must ensure that the Security Key Lifecycle Manager for z/OS Server certificate is connected to the Security Key Lifecycle Manager for z/OS's keyring. This example shows connecting the certificate that identifies the Security Key Lifecycle Manager for z/OS server to the Security Key Lifecycle Manager for z/OS keyring. Modify these command examples to suit your needs.

```
RACDCERT ID(ISKLMSRV) CONNECT(LABEL('ISKLMServer')RING(ISKLMRing))
```

d. If you are using ICSF, ensure that the Security Key Lifecycle Manager for z/OS Server instance has RACF authority to the key label of the private key stored in the ICSF PKDS. Also be sure to refresh the in-storage copies of the CSFKEYS Class profiles:

```
RDEFINE CSFKEYS ITOPS.ISKLM.CERT  UACC(NONE)
PERMIT ITOPS.ISKLM.CERT CLASS(CSFKEYS) ID(ISKLMSRV) ACCESS(READ)
SETROPTS RACLIST(CSFKEYS) GENERIC(CSFKEYS) REFRESH
```

2. **Generating a certificate signed by an Internal Certificate Authority**

a. Generate a self-signed certificate authority certificate.

```
RACDCERT CERTAUTH GENCERT SUBJECTSDN(CN('MyLocalzOSCA')O('MyCo')C('US'))
WITHLABEL('LocalRACF CA') PCICC(LOCAL.RACF.CERTAUTH) SIZE(2048)
```

If you are not using ICSF omit the PCICC keyword and change the keysize to 1024.

b. Generate an RSA key pair and certificate for the Security Key Lifecycle Manager for z/OS server instance on z/OS. The RACDCERT command illustrates that the certificate generated uses ICSF for private key storage. It is signed with the local certificate authority certificate generated in step 1.

```
RACDCERT ID(ISKLMSRV) GENCERT SUBJECTSDN(CN('ITOperations')
O('MyCo') C('US')) WITHLABEL('ISKLMServer') PCICC(ITOPS.ISKLM.CERT)
SIZE(2048) SIGNWITH(CERTAUTH LABEL('LocalRACF CA'))
```

If you are not using ICSF omit the PCICC keyword and change the keysize to 1024.

c. You can send this certificate to other business partners or sites within your enterprise. This is so that the certificate that identifies the Security Key Lifecycle Manager for z/OS instance on z/OS is known to your partners. Validate with your business partners or remote sites that you trust a common certificate authority (CA) whether third party or self signed. This depends on your business and security practices. This certificate can be imported into the keystore that is being used by the Security Key Lifecycle Manager for z/OS at the location of your partner. To send this certificate, you must export it to a dataset.

```
RACDCERT EXPORT (LABEL('ISKLMServer')) DSN('hlq.PUBKEY.S2048.ITOPS')
FORMAT(CERTDER)
```

See "Business Partner and Remote z/OS Systems " on page 58 for information.

d. Ensure that the Security Key Lifecycle Manager for z/OS Server certificate and its designated certificate authority certificate are connected to the key ring of the Security Key Lifecycle Manager for z/OS. These examples show connecting a certificate authority certificate. It also shows connecting the certificate that identifies the Security Key Lifecycle Manager for z/OS server to the Security Key Lifecycle Manager for z/OS keyring. Modify these command examples to suit your needs.

```
RACDCERT ID(ISKLMSRV) CONNECT(CERTAUTH LABEL('LocalRACF CA') RING(ISKLMRing))
RACDCERT ID(ISKLMSRV) CONNECT(LABEL('ISKLMServer')RING(ISKLMRing))
```

e. If you are using ICSF, ensure that the Security Key Lifecycle Manager for z/OS Server instance has RACF authority to the key label of the private key stored in the ICSF PKDS. Also be sure to refresh the in-storage copies of the CSFKEYS Class profiles:

```
RDEFINE CSFKEYS ITOPS.ISKLM.CERT  UACC(NONE)
PERMIT ITOPS.ISKLM.CERT CLASS(CSFKEYS) ID(ISKLMSRV) ACCESS(READ)
SETROPTS RACLIST(CSFKEYS) GENERIC(CSFKEYS) REFRESH
```

3. **Generating a certificate signed by a third-party certificate authority**

a. Generate an RSA key pair for the Security Key Lifecycle Manager for z/OS server instance on z/OS. The following RACDCERT command illustrates that the certificate generated uses ICSF for private key storage.

```
RACDCERT ID(ISKLMSRV) GENCERT SUBJECTSDN(CN('ITOperations')
O('MyCo') C('US')) WITHLABEL('ISKLMServer') PCICC(ITOPS.ISKLM.CERT) SIZE(2048)
```

If you are not using ICSF omit the PCICC keyword and change the keysize to 1024. The certificate can be submitted to a third-party certificate provider for signing.

b. Generate and save a certificate request to a dataset (hlq.PUBKEY.REQUEST.ITOPS)

```
RACDCERT GENREQ (LABEL('ISKLMServer')) DSN('hlq.PUBKEY.S2048.ITOPS')
```

c. Submit certificate request, hlq.PUBKEY.S2048.ITOPS to your certificate provider. The response you receive is an X.509 certificate. This example assumes that the certificate you receive from your third-party certificate authority is saved in the dataset 'hlq.THIRD.PARTY.CERT' on z/OS. The contents of this dataset is imported into RACF. This dataset contains only the signed certificate that identifies the Security Key Lifecycle Manager for z/OS instance running on z/OS. It can also contain the certificate authority certificate. The private key for the Security Key Lifecycle Manager for z/OS certificate remains protected by either ICSF in the PKDS or RACF.

d. Receive the response into dataset 'hlq.THIRD.PARTY.CERT'.

e. Add the certificate to RACF.

```
RACDCERT ADD('hlq.THIRD.PARTY.CERT') TRUST
WITHLABEL('ISKLMServer') ID(ISKLMSRV)
```

If the CA certificate is not contained in the dataset ‘hlq.THIRD.PARTY.CERT' you need to acquire the CA certificate that signed the Security Key Lifecycle Manager for z/OS certificate from the External certificate authority. You must then add it to RACF as a CERTAUTH.

f. You can send this certificate to other business partners or sites within your enterprise. This is so that the certificate that identifies the Security Key Lifecycle Manager for z/OS instance on z/OS is known to your partners. Validate with your business partners or remote sites that you trust a common certificate authority (CA) whether third-party or self signed. This depends on your business and security practices. This certificate can be

imported into the keystore that is being used by the Security Key Lifecycle Manager for z/OS at the location of your partner. To send this certificate, you must export it to a dataset

```
RACDCERT EXPORT (LABEL('ISKLMServer'))
DSN('hlq.PUBKEY.S2048.ITOPS') FORMAT(CERTDER)
```

See “Business Partner and Remote z/OS Systems ” for information.

g. Ensure that the Security Key Lifecycle Manager for z/OS Server certificate and its designated certificate authority certificate are connected to the key ring of the Security Key Lifecycle Manager for z/OS. These examples show connecting a certificate authority certificate. It also shows connecting the certificate that identifies the Security Key Lifecycle Manager for z/OS server to the Security Key Lifecycle Manager for z/OS keyring. Modify these command examples to suit your needs.

```
RACDCERT ID(ISKLMSRV) CONNECT(CERTAUTH LABEL('External CA label') RING(ISKLMRing))

RACDCERT ID(ISKLMSRV) CONNECT(LABEL('ISKLMServer')RING(ISKLMRing))
```

See “Business Partner and Remote z/OS Systems ” for information.

h. If you are using ICSF, ensure that the Security Key Lifecycle Manager for z/OS Server instance has RACF authority to the key label of the private key stored in the ICSF PKDS. Also be sure to refresh the in-storage copies of the CSFKEYS Class profiles:

```
RDEFINE CSFKEYS ITOPS.ISKLM.CERT  UACC(NONE)
PERMIT ITOPS.ISKLM.CERT CLASS(CSFKEYS) ID(ISKLMSRV) ACCESS(READ)
SETROPTS RACLIST(CSFKEYS) GENERIC(CSFKEYS) REFRESH
```

Business Partner and Remote z/OS Systems

Another z/OS business partner, for example, or perhaps a remote z/OS site within your business would import the certificate into a z/OS dataset. They then use the RACDCERT to add that certificate to RACF. The public key in the certificate can also be saved in the ICSF PKDS depending on the operands supplied to the RACDCERT command.

Note: The KeyLabel you specify on the WITHLABEL option of the **RACDCERT ADD** command must match the KeyLabel that was used when the certificate was created. This only works if you plan to specify an encoding mechanism of Label “L” when encrypting tapes. Optionally, you can specify an encoding mechanism of Public Key Hash “H”, which uses a Hash value rather than the KeyLabel to identify the key. While Hash gives slightly less performance, it allows you to import a certificate/public key without knowing the KeyLabel that was used to create/export the key. It also gives you the freedom to specify the KeyLabel you want to use to identify the public key of your business partner. Therefore using the Public Key Hash would be the preferred method. An example of how the z/OS encoding mechanism is specified in this section.

The following **RADCERT ADD** command imports the certificate and public key from the business partner contained in the 'dataset_containing_the_cert_received' with an alias CompanyXISKLMServer. In this example the imported alias of CompanyXISKLMServer does not match the original business partner's alias of ISKLMServer. This only works if you plan to specify an encoding mechanism of Public Key Hash “H” for this key as shown in the example. Otherwise the alias on the import command must be provided to you by your business partner.

```
RACDCERT ID(ISKLMSRV) ADD('dataset_containing_the_cert_received')
TRUST WITHLABEL('CompanyXISKLMServer') PCICC(companyX.ISKLMSRV.cert)
```

If you are not using ICSF omit the PCICC keyword and change the keysize to 1024.

The WITHLABEL keyword associates a string or *friendly name* for the certificate being imported. This name is used by Security Key Lifecycle Manager for z/OS when accessing the certificate. See the *z/OS Security Server RACF Command Language Reference* for detailed discussion of the **RACDCERT** command.

On z/OS the encoding key mechanism can be specified in the data class or JCL. This is an example of how it would be specified in the JCL. It is best that you specify an encoding mechanism of Label "L" when defining your own Key. You must specify an encoding mechanism of Public Key Hash "H" when defining a business partner key.

```
//C02STRW1 JOB CONSOLE,
// MSGCLASS=H,MSGLEVEL=(1,1),CLASS=B,
// TIME=1440,REGION=2M
/*JOBPARM SYSAFF=*
//*
//* ENC KEY MASTER JOB
//*
//CREATE1 EXEC PGM=IEBDG
//SYSPRINT DD SYSOUT=*
//SEQ001 DD DSN=TAPE.C02M5CX2.PC5.NOPOOL.C02STRS1.MASTER,
// KEYLABL1='ISKLMServer',
// KEYENCD1=L,
// KEYLABL2='CompanyXISKLMServer',
// KEYENCD2=H,
// LABEL=(1,SL),UNIT=C02M5CX2,DISP=(,CATLG),
// DCB=(DSORG=PS,RECFM=FB,LRECL=2048,BLKSIZE=6144)
//SYSIN DD *
DSD OUTPUT=(SEQ001)
FD NAME=A,STARTLOC=1,LENGTH=10,FORMAT=ZD,INDEX=1
FD NAME=B,STARTLOC=11,LENGTH=13,PICTURE=13,'PRIMER RECORD'
CREATE QUANTITY=25,FILL='Z',NAME=(A,B)
END
/*
```

For more information about the Hash encoding mechanism, see *z/OS DFSMS Software Support for IBM System Storage TS1130 and TS1120 Tape Drive (3592)*, SC26-7514.

You must ensure that this certificate is connected (or associated) with the keyring of the Security Key Lifecycle Manager for z/OS server. Use the **RACDCERT** command as shown in the example. This example assumes that the Security Key Lifecycle Manager for z/OS keyring on this z/OS system is ISKLMRing. It also assumes that the z/OS user ID associated with the Security Key Lifecycle Manager for z/OS process is ISKLMSRV.

Note: As this certificate contains only a public key, it is important that the USAGE(CERTAUTH) option is used. If it is not specified, the Security Key Lifecycle Manager for z/OS does not start. This is because it believes that the certificate that was added must also contain a private key.

```
RACDCERT ID(ISKLMSRV) CONNECT(LABEL('CompanyXISKLMServer')
RING(ISKLMRing) USAGE(CERTAUTH))

RACDCERT ID(ISKLMSRV) CONNECT(CERTAUTH LABEL('GENERATED CA Label FROM ADD')
RING(ISKLMRing))
```

On this *remote* z/OS system, ensure that the Security Key Lifecycle Manager for z/OS server is authorized to read from its keyring. It must also be authorized to use the key ICSF key label. Ensure that the required RACF FACILITY class profiles are defined. If not, issue the RDEFINE commands to define these profiles which protect the use of keyring functions:

```
RDEFINE FACILITY IRR.DIGTCERT.LIST UACC(NONE)
RDEFINE FACILITY IRR.DIGTCERT.LISTRING UACC(NONE)

PERMIT IRR.DIGTCERT.LIST CLASS(FACILITY) ID(ISKLMSRV) ACC(READ)
PERMIT IRR.DIGTCERT.LISTRING CLASS(FACILITY) ID(ISKLMSRV) ACC(READ)
```

If using ICSF, ensure that the Security Key Lifecycle Manager for z/OS Server instance has RACF authority to the key label of the private key stored in the ICSF PKDS. Also, issue refresh for the in-storage copies of the CSFKEYS Class profiles:

```
RDEFINE CSFKEYS REMOTE.ISKLM.CERT  UACC(NONE)
PERMIT REMOTE.ISKLM.CERT CLASS(CSFKEYS) ID(ISKLMSRV) ACCESS(READ)
SETROPTS RACLIST(CSFKEYS) GENERIC(CSFKEYS) REFRESH
```

For information about how to use this certificate containing the public key when encrypting data to tape, see *z/OS DFSMS Software Support for IBM System Storage TS1130 and TS1120 Tape Drives (3592)*, SC26-7514.

Verifying the Keyring can be accessed by the Java Keytool/Hwkeytool

Before starting the Security Key Lifecycle Manager for z/OS you must validate that the keystore is accessible in the Java hwkeytool (for JCECCARACFKS/ICSF keys) or Java keytool (for JCERACFKS/non-ICSF keys).

The examples show how you would use both **hwkeytool** and **keytool** to list the contents of the keyring.

For ICSF keys:

```
hwkeytool -debug -J-Djava.protocol.handler.pkgs=com.ibm.crypto.hdwrCCA.provider
-list -keystore safkeyring://ISKLMSRV/ISKLMRing -storetype JCECCARACFKS
```

For keys not in ICSF:

```
keytool -debug -J-Djava.protocol.handler.pkgs=com.ibm.crypto.provider
-list -keystore safkeyring://ISKLMSRV/ISKLMRing -storetype JCERACFKS
```

Creating Symmetric Keys for Use with LTO Ultrium 4 and LTO Ultrium 5 Drives

Although z/OS does not support LTO drives, you can run your Security Key Lifecycle Manager for z/OS and allow an off-platform LTO drive to retrieve keys from your z/OS. Your Security Key Lifecycle Manager for z/OS can service TS1120, TS1130, TS1140, LTO Ultrium 4 and LTO Ultrium 5 drives at the same time. However, for LTO Ultrium 4 and LTO Ultrium 5 drives you must manually create the symmetric keys in your Security Key Lifecycle Manager for z/OS keystore to be used for data encryption. See “How the Security Key Lifecycle Manager for z/OS Processes Encryption Keys ” on page 5 for an overview.

Symmetric keys are not supported by RACF. Your Security Key Lifecycle Manager for z/OS keystore must be of type JCEKS or JCECCAKS in order to create symmetric keys for use with LTO Ultrium 4 and LTO Ultrium 5 drives. The minimum required SDK installation for creating symmetric keys in a JCEKS type keystore is 50sr5. For JCECCAKS type keystore it is 50sr6.

For more information about using the Java **keytool**, see “Generating Keys and Aliases for Encryption on LTO Ultrium 4 and LTO Ultrium 5” on page 72.

Sample Alias and Symmetric Key Setup for LTO Ultrium 4 or LTO Ultrium 5 Drives Encryption

```
/u/giampor/tkms:>cat populatesymmkeys.jceccaks.sh
#
echo "Creating RSA Certificate and Public and Private KeyPair in isklm2sharedkeysjceccaksPKDSlabel"
#
keytool -genkey -alias symmkeywrapper -dname "CN=sharedkeysjceccaksSymmetricKeyWrapper" \
  -keystore isklm2sharedkeysjceccaksPKDSlabel -provider IBMJCECCA -keyalg RSA -keysize 2048 \
  -keypass "password" -storepass "password" -storetype JCECCAKS -validity 999
#
echo "List isklm2sharedkeysjceccaksPKDSlabel"
keytool -list -keystore isklm2sharedkeysjceccaksPKDSlabel -storepass "password" \
 -storetype JCECCAKS
#
#
echo "Exporting RSA Certificate/Public Key to isklm2sharedkeysjceccaksPKDSlabelCA.crt"
#
keytool -export -alias symmkeywrapper -file isklm2sharedkeysjceccaksPKDSlabelCA.crt \
  -keystore isklm2sharedkeysjceccaksPKDSlabel -provider IBMJCECCA -storepass "password" \
  -storetype JCECCAKS
#
#
echo "Creating Symmetric keys in symmkeystorejceccaks"
#
keytool -genseckey -keystore symmkeystorejceccaks -storetype JCECCAKS \
  -storepass "symmpassword" -aliasrange ibm01-05 -keyAlg DESede
#
echo "Listing Symmetric keys created"
#
keytool -list -keystore symmkeystorejceccaks -storepass "symmpassword" -storetype JCECCAKS
#
#
echo "import public keys from keystores who want a copy of symmetric keys using a different
 alias - in this case isklm2sharedkeysjceccaksPKDSlabelCA.crt from
 isklm2sharedkeysjceccaksPKDSlabel"
#
keytool -import -trustcacerts -alias sharedkeysjceccaksCA \
 -file isklm2sharedkeysjceccaksPKDSlabelCA.crt \
 -keystore symmkeystorejceccaks -storepass "symmpassword" -storetype JCECCAKS
#
#
echo "Listing keystore with public key imported and Symmetric keys"
#
keytool -list -keystore symmkeystorejceccaks -storepass "symmpassword" -storetype JCECCAKS
#
#
echo "Export the Symmetric keys from symmetrickeystore for
 sharedkeysjceccaksCA/isklm2sharedkeysjceccaksPKDSlabel"
#
keytool -exportseckey -aliasrange ibm01-05 -keyalias sharedkeysjceccaksCA \
 -keystore symmkeystorejceccaks \
 -storepass "symmpassword" -storetype JCECCAKS -keypass "symmpassword" \
 -exportfile symKeysexported.jcecca.cer
#
#
echo "Import the Symmetric keys into isklm2sharedkeysjceccaksPKDSlabel
  - i.e., sharedkeysjceccaksCA but assume must use my orignal alias symmkeywrapper
  or it won't know how to get the private key"
#
keytool -importseckey -keyalias symmkeywrapper -keypass "password" \
 -keystore isklm2sharedkeysjceccaksPKDSlabel \
 -storepass "password" -storetype JCECCAKS -importfile symKeysexported.jcecca.cer
#
#
echo "list isklm2keystore containing RSA keypair and symmetric keys"
#
keytool -list -keystore isklm2sharedkeysjceccaksPKDSlabel -storepass "password" \
 -storetype JCECCAKS
/u/giampor/tkms:>
```

Setting up the Security Key Lifecycle Manager for z/OS keystore to communicate with tape drives

The topic shows an example of a shell script you can create. You can create it to configure a JCECCAKS type keystore with 15 symmetric keys. It can be used to serve keys to TS1120, TS1130, TS1140, LTO Ultrium 4, and LTO Ultrium 5 tape drives.

```
/u/isklmsrv/temp/symmkeytest:>cat createisklmkeys.sh
#######################################
# Setup ISKLM Keystore
#######################################
echo "\nCreating the ISKLM Server Certificate and corresponding RSA Public and Private KeyPair in
 ISKLMKeystoreCCA where private key is stored in ICSF PKDS. \n"#
```

```
hwkeytool -genkey -alias CERT2 -dname "CN=MyCo ISKLM Server" -keystore ISKLMKeystoreCCA \
 -provider IBMJCECCA -keyalg RSA -keysize 2048 -keypass "password" -storepass "password" \
 -storetype JCECCAKS -hardwaretype PKDS -validity 999
#
echo "\nCreating a Certificate and correpsonding RSA Public and Private Keypair to be used to
 wrap symmetric keys that will be imported into ISKLMKeystoreCCA.
 Private key is also stored in ICSF PKDS. \n"
#
hwkeytool -genkey -alias symmkeywrapper -dname "CN=ISKLM Symmetric Key Wrapper" \
 -keystore ISKLMKeystoreCCA -provider IBMJCECCA -keyalg RSA -keysize 2048 -keypass "password" \
 -storepass "password" -storetype JCECCAKS -hardwaretype PKDS -validity 999
#
echo "\nList contents of the ISKLMKeystoreCCA. \n"
keytool -list -keystore ISKLMKeystoreCCA -storepass "password" -storetype JCECCAKS
#
echo "\nExporting the Symmetric Key Wrapper Certificate/Public Key to myCoSymmKeyWrapper.crt. \n"
#
keytool -export -alias symmkeywrapper -file myCoSymmKeyWrapper.crt -keystore ISKLMKeystoreCCA \
 -provider IBMJCECCA -storepass "password" -storetype JCECCAKS
#
################################################################################################
# Create symmetric keys
#
# This example creates 15 symmetric keys in a separate keystore and exports them to a
# file to be imported into the ISKLM Keystore.
# Optionally the symmetric keys can be created right into the ISKLM keystore.
# This example is to show the various symmetric/secure key commands.
# Note that only the keytool can be used (not hwkeytool), thus the there is no option to
# specify -hardwaretype PKDS to store the symmetric keys in ICSF but rather the symmetric
# keys will be stored in a password protected keystore.
# Also note that the -keyAlg must be DESede if specifying -storetype JCECCAKS or
# planning to exchange tapes with a z/OS ISKLM defined with that storetype.
# For DESede, the -keysize default is 168.
# For -storetype JCEKS, -keyAlg should be AES and -keysize 256
# (unless zOSCompatibility flag is set to true).
################################################################################################
#
echo "\nCreating 15 Symmetric keys in symmkeystorejceccaks. \n"
#
keytool -genseckey -keystore symmkeystorejceccaks -storetype JCECCAKS -storepass "symmpassword" \
 -keypass "symmpassword" -aliasrange ibm01-0F -keyAlg DESede
#
echo "\nListing Symmetric keys created. \n"
#
keytool -list  -keystore symmkeystorejceccaks -storepass "symmpassword" -storetype JCECCAKS
#
echo "\nImport public keys from keystores who want a copy of symmetric keys..in this case alias
 symmkeywrapper which was exported from EMKeystore4758 into myCoSymmKeyWrapper.crt.  Note that I
 can change the alias on import and that by specifying noprompt the certificate will be imported
 as a trusted certificate. \n"
#
keytool -import -noprompt -alias ExternalCoCA -file myCoSymmKeyWrapper.crt \
 -keystore symmkeystorejceccaks -storepass "symmpassword" -storetype JCECCAKS
#
echo "\nListing keystore with public key imported and Symmetric keys. \n"
#
keytool -list -keystore symmkeystorejceccaks -storepass "symmpassword" -storetype JCECCAKS
#
echo "\nExport the Symmetric keys from the symmetric keystore for ExternalCoCA
 (i.e., symmkeywrapper in ISKLMKeystoreCCA). \n"
#
keytool  -exportseckey -aliasrange ibm01-0F -keyalias ExternalCoCA \
 -keystore symmkeystorejceccaks -storepass "symmpassword" -storetype JCECCAKS \
 -keypass "symmpassword" -exportfile symKeysexported.ExternalCoCA.cer
#
################################################################################################
# Import the symmetric keys into the ISKLMKeystoreCCA.  Again note that there is currently no
# option to specify the keys to go into the ICSF PKDS but rather the keys will be stored in the
# password protected keystore.
################################################################################################
echo "\nImport the Symmetric keys into ISKLMKeystoreCCA.  Note that you must use the correct
 alias, in this case 'symmkeywrapper' or keytool won't know how to find the private key. \n"
#
keytool  -importseckey -keyalias symmkeywrapper -keypass "password" -keystore ISKLMKeystoreCCA \
 -storepass "password" -storetype JCECCAKS -importfile symKeysexported.ExternalCoCA.cer
#
echo "\nList ISKLMKeystoreCCA containing 2 RSA keypairs and 15 imported symmetric keys.\n"
#
keytool -list -keystore ISKLMKeystoreCCA  -storepass "password"  -storetype JCECCAKS
```

Sample Output from Shell Script

This section shows an example of the output of running the shell script:

```
/u/isklmsrv/temp/symmkeytest:>. createisklmkeys.sh

Creating the ISKLM Server Certificate and corresponding RSA Public and Private KeyPair in
ISKLMKeystoreCCA where private key is stored in ICSF PKDS.
```

```
Creating a Certificate and correpsonding RSA Public and Private Keypair to be used
to wrap symmetric keys that will be imported into ISKLMKeystoreCCA.  Private key
is also stored in ICSF PKDS.


List contents of the ISKLMKeystoreCCA.


Keystore type: JCECCAKS
Keystore provider: IBMJCECCA

Your keystore contains 2 entries

symmkeywrapper, Jul 31, 2007, keyEntry,
Certificate fingerprint (MD5): 89:E0:0A:32:A7:B4:A4:F1:F6:4D:B9:F5:68:69:91:C3
CERT2, Jul 31, 2007, keyEntry,
Certificate fingerprint (MD5): 16:B1:94:79:C1:C6:77:C5:6E:84:99:5C:D1:88:9E:65

Exporting the Symmetric Key Wrapper Certificate/Public Key to myCoSymmKeyWrapper.crt.

Certificate stored in file <myCoSymmKeyWrapper.crt>

Creating 15 Symmetric keys in symmkeystorejceccaks.

KeyTool is generating batch keys. This process will take a while, be patient ...
15 secret keys have been generated

Listing Symmetric keys created.


Keystore type: JCECCAKS
Keystore provider: IBMJCECCA

Your keystore contains 15 entries

ibm000000000000000008, Jul 31, 2007, keyEntry,
ibm00000000000000000f, Jul 31, 2007, keyEntry,
ibm000000000000000007, Jul 31, 2007, keyEntry,
ibm00000000000000000e, Jul 31, 2007, keyEntry,
ibm000000000000000006, Jul 31, 2007, keyEntry,
ibm00000000000000000d, Jul 31, 2007, keyEntry,
ibm000000000000000005, Jul 31, 2007, keyEntry,
ibm00000000000000000c, Jul 31, 2007, keyEntry,
ibm000000000000000004, Jul 31, 2007, keyEntry,
ibm00000000000000000b, Jul 31, 2007, keyEntry,
ibm000000000000000003, Jul 31, 2007, keyEntry,
ibm00000000000000000a, Jul 31, 2007, keyEntry,
ibm000000000000000002, Jul 31, 2007, keyEntry,
ibm000000000000000001, Jul 31, 2007, keyEntry,
ibm000000000000000009, Jul 31, 2007, keyEntry,

Import public keys from keystores who want a copy of symmetric keys..in this case alias
symmkeywrapper which was exported from EMKeystore4758 into myCoSymmKeyWrapper.crt.
Note that I can change the alias on import and that by specifying noprompt the certificate
will be imported as a trusted certificate.

Certificate was added to keystore

Listing keystore with public key imported and Symmetric keys.


Keystore type: JCECCAKS
Keystore provider: IBMJCECCA

Your keystore contains 16 entries

ibm000000000000000008, Jul 31, 2007, keyEntry,
ibm00000000000000000f, Jul 31, 2007, keyEntry,
ibm000000000000000007, Jul 31, 2007, keyEntry,
ibm00000000000000000e, Jul 31, 2007, keyEntry,
ibm000000000000000006, Jul 31, 2007, keyEntry,
ibm00000000000000000d, Jul 31, 2007, keyEntry,
ibm000000000000000005, Jul 31, 2007, keyEntry,
ibm00000000000000000c, Jul 31, 2007, keyEntry,
ibm000000000000000004, Jul 31, 2007, keyEntry,
ibm00000000000000000b, Jul 31, 2007, keyEntry,
ibm000000000000000003, Jul 31, 2007, keyEntry,
ibm00000000000000000a, Jul 31, 2007, keyEntry,
ibm000000000000000002, Jul 31, 2007, keyEntry,
ibm000000000000000001, Jul 31, 2007, keyEntry,
externalcoca, Jul 31, 2007, trustedCertEntry,
Certificate fingerprint (MD5): 89:E0:0A:32:A7:B4:A4:F1:F6:4D:B9:F5:68:69:91:C3
ibm000000000000000009, Jul 31, 2007, keyEntry,

Export the Symmetric keys from the symmetric keystore for ExternalCoCA
 (i.e., symmkeywrapper in ISKLMKeystoreCCA).

15 secret keys have been successfully exported

Import the Symmetric keys into ISKLMKeystoreCCA.  Note that you must use the correct alias,
```

```
 in this case 'symmkeywrapper' or keytool won't know how to find the private key.

15 secret keys have been imported

List ISKLMKeystoreCCA containing 2 RSA keypairs and 15 imported symmetric keys.


Keystore type: JCECCAKS
Keystore provider: IBMJCECCA

Your keystore contains 17 entries

ibm000000000000000008, Jul 31, 2007, keyEntry,
ibm00000000000000000f, Jul 31, 2007, keyEntry,
ibm000000000000000007, Jul 31, 2007, keyEntry,
ibm00000000000000000e, Jul 31, 2007, keyEntry,
ibm000000000000000006, Jul 31, 2007, keyEntry,
ibm00000000000000000d, Jul 31, 2007, keyEntry,
ibm000000000000000005, Jul 31, 2007, keyEntry,
ibm00000000000000000c, Jul 31, 2007, keyEntry,
ibm000000000000000004, Jul 31, 2007, keyEntry,
ibm00000000000000000b, Jul 31, 2007, keyEntry,
ibm000000000000000003, Jul 31, 2007, keyEntry,
ibm00000000000000000a, Jul 31, 2007, keyEntry,
ibm000000000000000002, Jul 31, 2007, keyEntry,
ibm000000000000000001, Jul 31, 2007, keyEntry,
symmkeywrapper, Jul 31, 2007, keyEntry,
Certificate fingerprint (MD5): 89:E0:0A:32:A7:B4:A4:F1:F6:4D:B9:F5:68:69:91:C3
CERT2, Jul 31, 2007, keyEntry,
Certificate fingerprint (MD5): 16:B1:94:79:C1:C6:77:C5:6E:84:99:5C:D1:88:9E:65
ibm000000000000000009, Jul 31, 2007, keyEntry,
```

Setting up the Security Key Lifecycle Manager for z/OS configuration file

Before you start Security Key Lifecycle Manager for z/OS, you must create the configuration file for your Security Key Lifecycle Manager for z/OS. The configuration file defines many parameters including which port the Security Key Lifecycle Manager for z/OS runs on. It also defines the keystore or keyring where it can find the X.509 Digital Certificates. See Chapter 4, “Configuring the Security Key Lifecycle Manager for z/OS,” on page 79.

Creating file system and mount point to contain configuration files

The configuration file is initially read by the Security Key Lifecycle Manager for z/OS server upon startup. The file is written back to the file when the Security Key Lifecycle Manager for z/OS server is stopped. During the startup process, the Security Key Lifecycle Manager for z/OS attempts a filecreate to the directory for a file with the name **.backup**. This setting is done to ensure that the server can write back to the configuration file when stopped.

Take these steps:

1. Create a file system, for example **hlq.ISKLMSRV.ZFS** and mount at **/u/isklmsrv**. See sample job SCKLSAMP(CKLCZFS).
2. Define a directory for each system to contain a Security Key Lifecycle Manager for z/OS configuration file unique to that specific system.
3. Ensure the RACF ID that the Security Key Lifecycle Manager for z/OS server uses, ISKLMSRV, is the owner of the **/u/isklmsrv** and subsequent directories and files.

The following are samples of configuration files for each keystore type, which can help you to get started. The examples show various keystore locations, directories, and file names. You must customize them for your system configuration. The examples for JCECCARACFKS and JCERACFKS show setting up a system named JA0. It is a member of a sysplex that is setup to use Unix Systems Services Shared

HFS. A sample configuration file can be found in /usr/lpp/ISKLM/samples/ISKLMConfig.properties.zos.

Example Configuration File for JCEKS

This example configuration file for JCEKS is named **/u/isklmsrv/ISKLMConfig.properties.zos.jceks**.

```
Admin.ssl.keystore.name = /u/isklmsrv/ISKLMKeystore
Admin.ssl.keystore.password = password
Admin.ssl.keystore.type = jceks
Admin.ssl.truststore.name = /u/isklmsrv/ISKLMKeystore
Audit.event.outcome = success,failure
Audit.event.types = all
Audit.eventQueue.max = 0
Audit.handler.file.directory = /u/isklmsrv/keylifecyclemanager/keylifecyclemanager/audit
Audit.handler.file.name = isklmaudit.log.jceks
Audit.handler.file.size = 10000
Audit.metadata.file.name = /u/isklmsrv/metafile.xml
config.drivetable.file.url = FILE:///u/isklmsrv/keylifecyclemanager/drivetable
config.keystore.file = /u/isklmsrv/ISKLMKeystore
config.keystore.password = password
config.keystore.provider = IBMJCE
config.keystore.type = jceks
drive.acceptUnknownDrives = true
ds8k.acceptUnknownDrives = true
drive.default.alias1 = CERT1
drive.default.alias2 = CERT1
fips = Off
TransportListener.ssl.ciphersuites = JSSE_ALL
TransportListener.ssl.clientauthentication = 0
TransportListener.ssl.keystore.name = /u/isklmsrv/ISKLMKeystore
TransportListener.ssl.keystore.password = password
TransportListener.ssl.keystore.type = jceks
TransportListener.ssl.port = 5443
TransportListener.ssl.protocols = SSL_TLS
TransportListener.ssl.truststore.name = /u/isklmsrv/ISKLMKeystore
TransportListener.ssl.truststore.type = jceks
TransportListener.tcp.port = 3801
```

If supporting LTO drives, add

```
symmetricKeySet = ibm01-1F4
```

This would represent 500 symmetric keys. Symmetric keys are only required when supporting LTO encryption drives.

Example Configuration File for JCECCARACFKS

This example configuration file for JCECCARACFKS is named **/u/isklmsrv/JA0/ISKLMConfig.properties.zos.JCECCARACFKS** .

```
Admin.ssl.keystore.name = safkeyring://ISKLMSRV/ISKLMRing
Admin.ssl.truststore.name = safkeyring://ISKLMSRV/ISKLMRing
Audit.event.outcome = success,failure
Audit.event.types = all
Audit.eventQueue.max = 0
Audit.handler.file.directory = /isklmlogs/JA0/audit
Audit.handler.file.name = audit.log
Audit.handler.file.size = 10000
Audit.metadata.file.name = /u/isklmsrv/metafile.xml
config.drivetable.file.url = FILE:/u/isklmsrv/JA0/filedrive.table
config.keystore.file = safkeyring://ISKLMSRV/ISKLMRing
config.keystore.password = password
config.keystore.provider = IBMJCECCA
config.keystore.type = JCECCARACFKS
drive.acceptUnknownDrives = true
ds8k.acceptUnknownDrives = true
drive.default.alias1 = ISKLMServer
drive.default.alias2 = ISKLMServer
fips = Off
requireHardwareProtectionForSymmetricKeys = true
TransportListener.ssl.ciphersuites = JSSE_ALL
TransportListener.ssl.clientauthentication = 0
TransportListener.ssl.keystore.name = safkeyring://ISKLMSRV/ISKLMRing
TransportListener.ssl.keystore.password = password
TransportListener.ssl.keystore.type = JCECCARACFKS
TransportListener.ssl.port = 1443
TransportListener.ssl.protocols = SSL_TLS
TransportListener.ssl.truststore.name = safkeyring://ISKLMSRV/ISKLMRing
TransportListener.ssl.truststore.type = JCECCARACFKS
TransportListener.tcp.port = 3801
```

Example Configuration File for JCERACFKS

This example configuration file for JCERACFKS is named **/u/isklmsrv/JA0/ISKLMConfig.properties.zos.JCERACFKS** .

```
Admin.ssl.keystore.name = safkeyring://ISKLMSRV/ISKLMRing
Admin.ssl.truststore.name = safkeyring://ISKLMSRV/ISKLMRing
Audit.event.outcome = success,failure
Audit.event.types = all
Audit.eventQueue.max = 0
Audit.handler.file.directory = /isklmlogs/JA0/audit
Audit.handler.file.name = audit.log
Audit.handler.file.size = 10000
Audit.metadata.file.name = /u/isklmsrv/metafile.xml
config.drivetable.file.url = FILE:/u/isklmsrv/JA0/filedrive.table
config.keystore.file = safkeyring://ISKLMSRV/ISKLMRing
config.keystore.password = password
config.keystore.provider = IBMJCE
config.keystore.type = JCERACFKS
drive.acceptUnknownDrives = true
ds8k.acceptUnknownDrives = true
drive.default.alias1 = ISKLMServer
drive.default.alias2 = ISKLMServer
fips = Off
TransportListener.ssl.ciphersuites = JSSE_ALL
TransportListener.ssl.clientauthentication = 0
TransportListener.ssl.keystore.name = safkeyring://ISKLMSRV/ISKLMRing
TransportListener.ssl.keystore.password = password
TransportListener.ssl.keystore.type = JCERACFKS
TransportListener.ssl.port = 1443
TransportListener.ssl.protocols = SSL_TLS
TransportListener.ssl.truststore.name = safkeyring://ISKLMSRV/ISKLMRing
TransportListener.ssl.truststore.type = JCERACFKS
TransportListener.tcp.port = 3801
```

Quick Test Running Security Key Lifecycle Manager for z/OS Under USS

For test environments you can start the Security Key Lifecycle Manager for z/OS from USS or OMVS to run in the foreground. Using this setting, you can quickly attempt to write or read an encrypted tape. However, for production purposes it is best that you follow the steps in the topic “Setting up and running Security Key Lifecycle Manager for z/OS in Production Mode” on page 67.

To run the Security Key Lifecycle Manager for z/OS in the foreground, you must have either a USS or OMVS session. You must verify the Java version in your path, see "Verify the Java Version" on page 44.

Ensure that you add the /usr/lpp/ISKLM/IBMSKLM.jar to your classpath. From your USS or OMVS session, you can run one of the following commands based on the keystore type you are using. Be sure to replace the properties file with the location of your properties file:

JCEKS

```
/u/isklmsrv/:>  java com.ibm.ltklm.ISKLMServer
/u/isklmsrv/ISKLMConfig.properties.zos.jceks
```

JCECCAKS

```
/u/isklmsrv/:>  java com.ibm.ltklm.ISKLMServer
/u/isklmsrv/ISKLMConfig.properties.zos.jceccaks
```

JCERACFKS

```
u/isklmsrv/:>
java -Djava.protocol.handler.pkgs=com.ibm.crypto.provider
com.ibm.ltklm.ISKLMServer
/u/isklmsrv/ISKLMConfig.properties.zos.jceracfks
```

JCECCARACFKS

```
u/isklmsrv/:>
java -Djava.protocol.handler.pkgs=com.ibm.crypto.hdwrCCA.provider
com.ibm.ltklm.ISKLMServer
/u/isklmsrv/ISKLMConfig.properties.zos.jceccaracfks
```

You can now attempt to write and read encrypted tapes using a tape drive against this Security Key Lifecycle Manager for z/OS Server. If you are setting up a z/OS tape drive to run in-band with your Security Key Lifecycle Manager for z/OS, see "Note about z/OS configuration steps for z/OS in-band encrypted tape drive" on page 88.

To stop the Security Key Lifecycle Manager for z/OS server submit the quit command. Shutting down the Security Key Lifecycle Manager for z/OS server overwrites the ISKLMConfig.properties.zos.jceks with any changes made to it while the Security Key Lifecycle Manager for z/OS Server is running. See Chapter 5, "Administering the Security Key Lifecycle Manager for z/OS," on page 89 for more information about Security Key Lifecycle Manager for z/OS commands.

```
# quit
Stopping the ISKLM admin service...
/u/isklmsrv:>
```

Setting up and running Security Key Lifecycle Manager for z/OS in Production Mode

Define the Security Key Lifecycle Manager for z/OS as a started task

A procedure consists of a set of job control language statements that are frequently used together to achieve a certain result. Procedures are located in the system procedure library, SYS1.PROCLIB, which is a partitioned data set. A started procedure is started by an operator, but can be associated with a functional subsystem.

Run Security Key Lifecycle Manager for z/OS as a started procedure on z/OS using the JZOS batch launcher. The JZOS batch launcher is shipped as part of the z/OS Java product. To define the Security Key Lifecycle Manager for z/OS as a started procedure update the started class table with the z/OS user ID of the Security Key Lifecycle Manager for z/OS. For more information about RACF processing and the definition of started procedures see, *z/OS Security Server RACF Security Administrator's Guide*.

1. In this example, the user ID of the Security Key Lifecycle Manager for z/OS instance on z/OS is ISKLMSRV. The group associated with the started procedure is sys1. You can tailor these examples to suit your needs. To set up the STARTED class, enter these commands (for example):

   ```
   SETROPTS GENERIC(STARTED)
   RDEFINE STARTED ISKLM*.* STDATA(USER(ISKLMSRV) GROUP(STCGROUP) TRACE(YES))
   SETROPTS CLASSACT(STARTED) SETROPTS RACLIST(STARTED)
   SETROPTS RACLIST(STARTED) GENERIC(STARTED) REFRESH
   ```

2. Create a home directory. Use the home directory specified on the RACF **adduser** command issued in "Setting up a user ID to run the Security Key Lifecycle Manager for z/OS" on page 45:

   ```
   Example:   /u/isklmsrv
   ```

Create a file system and mount point for Security Key Lifecycle Manager logging of debug and audit logs

The Security Key Lifecycle Manager for z/OS can create audit records that wrap the log to three files. When the last file becomes full, the first file is rewritten. The Chapter 7, "Audit Records," on page 117 topic explains the events that the Security Key Lifecycle Manager for z/OS audits and the format of the audit records. You must allocate a file system space for the Security Key Lifecycle Manager for z/OS audit logs. If requested by the IBM Service, the Security Key Lifecycle Manager for z/OS debug log might need to be enabled.

If the file system fills up and can no longer be extended, the Security Key Lifecycle Manager for z/OS continues to run without logging. However, a noticeable performance degradation can be encountered if the file system is an HFS. If you use a ZFS file system, there might not be a change in performance.

1. Allocate a file system specifically for use by the Security Key Lifecycle Manager for z/OS for audit and debug file storage. Assume 500 cylinders of space allocated to the Security Key Lifecycle Manager for z/OS's audit and debug log file system. Observe the tape and Security Key Lifecycle Manager for z/OS activity to determine how quickly the audit logs wrap. The file system must not be shared by the Security Key Lifecycle Manager for z/OS instances running in a sysplex environment. The file system must be private to each Security Key Lifecycle Manager for z/OS instance. This setting prevents any possible interleaving of audit or debug logs between Security Key Lifecycle Manager for z/OS instances.
2. Mount the isklmlogs file system and create a directory for each system that the Security Key Lifecycle Manager for z/OS runs on. For example, the two file systems created by the statements below are isklmlogs with JA0 and JB0 being two system names of two images within a sysplex.

   ```
   /isklmlogs/JA0
   /isklmlogs/JB0
   ```

Create a new PDS to contain the shell script for the JZOS launcher

1. Create a new PDS to contain the STDENV environmental variables. In this example, a portioned data set was allocated whose name is ISKLMSRV.ENCRYPT.CONFIG

```
ISKLMSRV.ENCRYPT.CONFIG
 Organization  . . . : PO
 Record format . . . : FB
 Record length . . . : 80
 Block size  . . . . : 6160
 1st extent cylinders: 3
     Secondary cylinders : 1
```

Create a Member in ISKLMSRV.ENCRYPT.CONFIG

Create/edit the shell script contents. See member CKLENV with alias ISKLMENV in the SCKLSAMP library. The text that is preceded by # in the example designates a comment. The example shows a setup for the system JA0. JA0 is a member of a PDS "ISKLMSRV.ENCRYPT.CONFIG" that is pointed to by the Security Key Lifecycle Manager for z/OS start procedure.

Shell script example to create member in ISKLMSRV.ENCRYPT.CONFIG

```
# This is a shell script which configures
# any environment variables for the Java JVM.
# Variables must be exported to be seen by the launcher.

. /etc/profile

export JAVA_HOME="/usr/lpp/java/J5.0"
export PATH=/bin:"${JAVA_HOME}"/bin:

LIBPATH=/lib:/usr/lib:"${JAVA_HOME}"/bin
LIBPATH="$LIBPATH":"${JAVA_HOME}"/bin/classic

export LIBPATH="$LIBPATH":

# Customize your CLASSPATH here
CLASSPATH=${JAVA_HOME}/lib
CLASSPATH=/u/isklmsrv:$CLASSPATH

export CLASSPATH="$CLASSPATH":

# Set JZOS specific options
export ISKLMCLASS="com.ibm.ltklm.ISKLMServer"
export ISKLMARGS="/u/isklmsrv/JA0/ISKLMConfig.properties.zos.jceccaracfks"
export JZOS_MAIN_ARGS="$ISKLMCLASS $ISKLMARGS"

# Configure JVM options (if any)
# for JCECCARACFKS, following IJO definition is required
IJO="-Djava.protocol.handler.pkgs=com.ibm.crypto.hdwrCCA.provider"
# for JCERACFKS, following IJO definition is required
# IJO="-Djava.protocol.handler.pkgs=com.ibm.crypto.provider"
# for JCEKS and JCECCAKS, no IJO definition is required
export IBM_JAVA_OPTIONS="$IJO"

#export JAVA_DUMP_HEAP=false
#export JAVA_PROPAGATE=NO
#export IBM_JAVA_ZOS_TDUMP=NO
```

Customize and install Security Key Lifecycle Manager start procedure

Create the started task JCL. See member CKLPROC with alias ISKLM in the SCKLSAMP library. Ensure that you review the installation documentation for the

z/OS Java product. The documentation contains additional guidelines, annotated samples, and step by step installation instructions for the JZOS Batch Launcher function.

The JZOS Batch Launcher is supported by its own set of environment variables. For further information about the JZOS Batch Launcher, see *JZOS Batch Launcher and Toolkit Installation and User's Guide* located at http://www.ibm.com/servers/eserver/zseries/software/java/jzos/overview.html.

JCL example to create member in a library in the system proclib concatenation.

Example of Security Key Lifecycle Manager for z/OS start procedure for z/OS

```
//ISKLM PROC JAVACLS='com.ibm.jzosekm.ISKLMConsoleWrapper',
//   ARGS=,                              < Args to Java class
//   LIBRARY='JZOS.LOADLIB',             < STEPLIB FOR JVMLDM module
//   VERSION='14',                       < JVMLDM version: 14, 50, 56
//    LOGLVL='+T',                       < Debug LVL: +I(info) +T(trc)
//   REGSIZE='0M',                       < EXECUTION REGION SIZE
//    LEPARM=''
//*****************************************************
//*ISKLM PROC JAVACLS='ISKLMConsoleWrapper', < fully qualified Java class
//*
//*Stored procedure for executing the JZOS Batch Launcher
//*Specifically, to execute the Security Key Lifecycle Manager under JZOS
//*
//*****************************************************
//ISKLM  EXEC PGM=JVMLDM&VERSION,REGION=&REGSIZE,
//     PARM='&LEPARM/&LOGLVL &JAVACLS &ARGS'
//STEPLIB  DD DSN=&LIBRARY,DISP=SHR
//SYSPRINT DD SYSOUT=*          < System stdout
//SYSOUT   DD SYSOUT=*          < System stderr
//STDOUT   DD SYSOUT=*          < Java System.out
//STDERR   DD SYSOUT=*          < Java System.err
//CEEDUMP  DD SYSOUT=*
//ABNLIGNR DD DUMMY
//*********************************************************************
//*
//*The following member contains the JVM environment script
//*
//* &SYSNAME symbolic for the system unique ConfigFile
//*
 //*******************************************************************
//STDENV DD DSN=ISKLMSRV.ENCRYPT.CONFIG(&SYSNAME.),DISP=SHR
//
```

Create Security Key Lifecycle Manager for z/OS configuration file

You must create an Security Key Lifecycle Manager for z/OS configuration file for each system on which you plan to run an Security Key Lifecycle Manager for z/OS. In this example, the configuration file is called **/u/isklmsrv/JA0/ISKLMConfig.properties.zos.jceccaracfks** where JAO is the system name. Read Chapter 4, "Configuring the Security Key Lifecycle Manager for z/OS," on page 79 for information about how to set up this file for z/OS production mode.

Starting and Stopping Security Key Lifecycle Manager on z/OS

The Security Key Lifecycle Manager for z/OS process can now be started with the operator **start** command, as a started task. The operator console can be used to issue commands through operator modify commands. The sample excerpt from the z/OS operators console, shows the start of the Security Key Lifecycle Manager for z/OS. This command is an operator issued modify command to list the drives known to the Security Key Lifecycle Manager for z/OS, and its termination.

Example contents of z/OS operator console

```
S ISKLM
$HASP100 ISKLM ON STCINRDR
IEF695I START ISKLM WITH JOBNAME ISKLM IS ASSIGNED TO USER ISKLMSRV, GROUP SYS1
$HASP373 ISKLM      STARTED
ISKLM console interaction is now available. 546
To submit commands to the ISKLM from the console:
    F ISKLM,APPL='ISKLM command'
To stop the ISKLM properly:
    P ISKLM
Loaded drive key store successfully
Starting the Encryption Key Manager 2.0-20070419
Processing Arguments
Processing
Server is started
Server is running. TCP port: 3801, SSL port: 443

F ISKLM,APPL='LISTDRIVES'
Drive entries: 14
SerialNumber = 00000AZ00011
SerialNumber = 000001365050
SerialNumber = 000001365043
SerialNumber = 000001365042
SerialNumber = 000001365041
SerialNumber = 000001365012
SerialNumber = 000001365067
SerialNumber = 000001365037
SerialNumber = 00000AZ00127
SerialNumber = 000001350808
SerialNumber = 00000AZ00125
SerialNumber = 000001365089
SerialNumber = 000001365088
SerialNumber = 000001365031

 P ISKLM
 Stopping the ISKLM admin service...
 Server is not started
 ISKLM stop command received
 IEF170I 1 ISKLM      - ------------------------------------------------
 IEF170I 1 ISKLM      -                                      REGION
 IEF170I 1 ISKLM      - STEPNAME PROCSTEP PGMNAME     CC     USED      CP
 IEF170I 1 ISKLM      -          ISKLM      JVMLDM14    00     100K   00:00
 IEF170I 1 ISKLM      - ------------------------------------------------
 IEF170I 1 ISKLM      - NAME-                    TOTALS: CPU TIME=  00:0
 IEF170I 1 ISKLM      - ------------------------------------------------
 $HASP395 ISKLM      ENDED
```

Note:
To properly end the Security Key Lifecycle Manager for z/OS server, issue the MVS™ STOP Command,

```
STOP ISKLM
```

from the operator console. If the Security Key Lifecycle Manager for z/OS is canceled (**CANCEL** command) instead of being stopped, then critical data can be lost.

In general, commands that can be issued from the shell environment to manage the Security Key Lifecycle Manager for z/OS can be issued from the operator console. These commands are specified using the modify command (F), with the syntax used for Security Key Lifecycle Manager for z/OS as documented in this publication enclosed in quotation marks. For example, use the following syntax:`f isklm,appl='isklm_command'`. If the isklm_command is not bounded by quotation marks, the command is not accepted by ISKLMConsoleWrapper and an error message appears on the console. Invalid isklm_commands bounded by quotation marks are not accepted by the Security Key Lifecycle Manager for z/OS server and

an error message appears on the console. Consult “Command Line Interface Commands” on page 92 for the valid Security Key Lifecycle Manager for z/OS command syntax.

Generating Keys and Aliases for Encryption on LTO Ultrium 4 and LTO Ultrium 5

Keytool is the preferred utility for managing keys, certificates, and aliases. It enables you to generate, import, and export your encryption data keys and store them in a keystore.

Each data key in the keystore is accessed through a unique alias. An alias is a string of characters, such as `123456tape`. In JCEKS and most other keystores, `123456Tape` would be equivalent to `123456tape` and can access the same entry in the keystore. Check the documentation for your keystore type to determine whether it is case-sensitive. When you use the **keytool -genseckey** command to generate a data key, you specify a corresponding alias in the same command. The alias identifies the correct key in the correct key group and keystore. It is used in writing and reading encrypted data on LTO Ultrium 4 and LTO Ultrium 5 tape.

Note: Individual aliases and alias ranges must be unique. This requirement is enforced when keys are generated on a given keystore/Security Key Lifecycle Manager for z/OS instance. However, in a multiple Security Key Lifecycle Manager for z/OS or Keystore environment, you should use a naming convention. Use a naming convention that maintains uniqueness across multiple instances in the event it becomes necessary to transport keys between instances while maintaining uniqueness of reference.

After generating keys and aliases, update the symmetricKeySet property in the ISKLMConfig.properties.zos file to specify the new alias, range of aliases, or key group GroupID. You must also update the filename under which the symmetric keys are stored, and the filename where key groups are defined. See “Creating and managing key groups” on page 76 for details. Only those keys named in the symmetricKeySet are validated (checked for an existing alias and a symmetric key of the proper size and algorithm). If an invalid key is specified in this property, the software does not start and an audit record is created.

The keytool utility also provides for the import and export of data keys to and from other keystores. An overview of each task follows. You can issue the **keytool -help** to show all the related parameters covered in the following discussions.

If you are not using Keytool

If you use a utility other than keytool to generate keys and aliases, you cannot generate ranges of keys compatible with the Security Key Lifecycle Manager for z/OS. To generate individual keys compatible with the Security Key Lifecycle Manager for z/OS, be sure to specify aliases using one of the following formats:

- 12 printable characters or less (for example, `abcdefghijk`)
- 3 printable characters, followed by two zeros, followed by 16 hexadecimal digits (for example, `ABC00000000000000001`) for a total of exactly 21 characters

Generating data keys and aliases using Keytool -genseckey

The Keytool utility generates aliases and symmetric keys for encryption on LTO Ultrium 4 and LTO Ultrium 5 Tape Drives using LTO Ultrium 4 and LTO Ultrium

5 tape. Use the **keytool -genseckey** command to generate one or more secret keys and store them in a specified keystore. **keytool -genseckey** takes the following parameters:

```
-genseckey      [-v] [-protected]
                [-alias <alias> | aliasrange <aliasRange>] [-keypass <keypass>]
                [-keyalg <keyalg>] [-keysize <keysize>]
                [-keystore <keystore>] [-storepass <storepass>]
                [-storetype <storetype>] [-providerName <name>]
                [-providerPath <pathlist>]
```

These parameters are important when generating data keys for Security Key Lifecycle Manager for z/OS to serve to the LTO Ultrium 4 and LTO Ultrium 5 drives for tape encryption:

-alias
Specify an *alias* value for a single data key with up to 12 printable characters (for example, `abcfrg` or `key123tape`).

-aliasrange
When generating multiple data keys, *aliasrange* is specified as a 3-character alphabetic prefix. It is followed by lower and upper limits for a series of 16-character (hexadecimal) strings with leading zeroes filledin automatically to construct aliases 21-characters in length. For example, specifying `key1-a` would yield a series of aliases from `KEY000000000000000001` through `KEY00000000000000000A`. Specifying an *aliasrange* value of `xyz01-FF` would yield `XYZ000000000000000001` through `XYZ0000000000000000FF`, which would generate 255 symmetric keys.

-keypass
Specifies a password used to protect the data key. This password **must be identical** to the keystore password. If no password is specified, you are prompted for it. If you press **Enter** at the prompt, the key password is set to the same password used for the keystore. *keypass* must be at least six characters long.

Note: When you have set the keystore password, **do not change** it unless its security has been breached. See "Changing Keystore Passwords" on page 74.

-keyalg
Specifies the algorithm to be used to generate the data key. If the encrypted tape is shared with systems on the z/OS platform and the zOSCompatibility property is set to true, the key algorithm should be specified as `DESede`. This setting ensures compatibility.

-keysize
Specifies the size of the data key to be generated. The key size must be specified as `256`. If -keyalg is specified as `DESede` for z/OS compatibility, then -keysize must be allowed to default to 168.

Examples of acceptable aliases that can be associated with symmetric keys are:

```
abc000000000000000001
abc00a0120fa000000001
```

Examples of aliases that would not be accepted are:

```
abcefghij1234567 ? wrong length
abcg0000000000000001 ? prefix is longer than 3 characters
```

If an alias exists in the keystore, keytool throws an exception and stops.

Changing Keystore Passwords

Note: When you have set the keystore password, **do not change** it unless its security has been breached. The passwords are obfuscated to eliminate any security exposure. This command will change both the keystore password and all of the aliases in the keystore with one command:

```
keytool -keystore <keystorename>  -storetype <keystoretype>
-keypasswd -new <newpassword> -all -storepasswd -new <newpassword>
```

Note: The value for both -new keywords MUST be identical.

You must also edit ISKLMConfig.properties.zos to change the keystore password in every server configuration file property where it is specified using one of these methods:

- Delete the entire obfuscated password and type the new password in the clear. This refers to the non-obfuscated property. For example, config.keystore.password. It will be obfuscated on the next startup.
- Delete the entire obfuscated password and set the Security Key Lifecycle Manager for z/OS to prompt on the next startup.

Importing data keys using Keytool -importseckey

Use the keytool -importseckey command to import a secret key or a batch of secret keys from an import file. **keytool -importseckey** takes the following parameters:

```
-importseckey          [-v]
               [-keyalias <keyalias>] [-keypass <keypass>]
               [-keystore <keystore>] [-storepass <storepass>]
               [-storetype <storetype>] [-providerName <name>]
               [-importfile <importfile>]
```

These parameters are important when importing data keys for the Security Key Lifecycle Manager for z/OS to serve to the LTO Ultrium 4 and LTO Ultrium 5 drives for tape encryption:

-keyalias
Specifies the alias of a private key in keystore to decrypt all the data keys in *importfile*.

-importfile
Specifies the file that contains the data keys to be imported.

Exporting data keys using keytool -exportseckey

Use the keytool -exportseckey command to export a secret key or a batch of secret keys to an export file. **keytool -exportseckey** takes the following parameters:

```
-exportseckey          [-v]
               [-alias <alias> | aliasrange <aliasRange>] [-keyalias <keyalias>]
               [-keystore <keystore>] [-storepass <storepass>]
               [-storetype <storetype>] [-providerName <name>]
               [-exportfile <exportfile>]
```

These parameters are important when exporting data keys for Security Key Lifecycle Manager for z/OS to serve to the LTO Ultrium 4 and LTO Ultrium 5 drives for tape encryption:

-alias
Specify an *alias* value for a single data key with up to 12 printable characters (for example, `abcfrg` or `key123tape`).

-aliasrange
When exporting multiple data keys, *aliasrange* is specified as a 3-character alphabetic prefix. It is then followed by lower and upper limits for a series of 16-character (hexadecimal) strings with leading zeroes filled-in automatically to construct aliases 21-characters in length. For example, specifying `key1-a` would yield a series of aliases from `KEY000000000000000001` through `KEY00000000000000000A`. Specifying an *aliasrange* value of `xyz01-FF` would yield `XYZ000000000000000001` through `XYZ0000000000000000FF`

-exportfile
Specifies the file to store the data keys when they are exported.

-keyalias
Specifies the alias of a public key in keystore to encrypt all the data keys. Ensure that the keystore where the symmetric (data) keys are, contains the corresponding private key.

Sample alias and symmetric key setup for LTO Ultrium 4 and LTO Ultrium 5 encryption using a JCEKS keystore

Invoke the **KeyTool** with the `-aliasrange` option.

If the encrypted tape is shared with systems on the z/OS platform and zOSCompatibility property is set to true, then key algorithm (-keyalg) must be specified as: `DESede`.

```
keytool –genseckey –v –aliasrange AES01-FF –keyalg DESede
–keypass password -storetype jceks –keystore path/filename.jceks
```

These KeyTool invocations generate 255 sequential aliases in the range AES000000000000000001 through AES0000000000000000FF and associated AES 256-bit symmetric keys. Either can be repeated cumulatively. It can be done as many times as necessary to setup the full number of ranged and stand-alone key aliases that are needed for robust operation. For example, to generate an additional alias and symmetric key for LTO Ultrium 4 and LTO Ultrium 5:

```
$JAVA_HOME/bin/keytool –genseckey –v –alias abcfrg –keyalg AES –keysize 256
–keypass password -storetype jceks –keystore path/filename.jceks
```

This invocation adds stand-alone alias `abcfrg` cumulatively. It is added to the named keystore which already contains 255 aliases from the invocation above yielding 256 symmetric keys in the JCEKS file named in –keystore option.

Update the symmetricKeySet property in the ISKLMConfig.properties.zos file. Add the following line to match any or all of the alias ranges used above, and the filename under which the symmetric keys were stored. The Security Key Lifecycle Manager for z/OS cannot start if an invalid alias is specified. Other causes for validation check failure can include incorrect bit size (for AES keysize must be 256) or an invalid algorithm for the platform. The file name specified in the **config.keystore.file** must match the name specified in the –keystore <filename> in the KeyTool invocation:

symmetricKeySet = AES01-FF,abcfrg
config.keystore.file = <filename>.jceks

Only those keys named in the symmetricKeySet is checked for an existing alias and a symmetric key of the proper size and algorithm. If an invalid key is specified in this property, the Security Key Lifecycle Manager for z/OS cannot start and an audit record is created.

Creating and managing key groups

The Security Key Lifecycle Manager for z/OS gives you the ability to organize your symmetric keys for LTO Ultrium 4 and LTO Ultrium 5 encryption into key groups. You can group keys according to the type of data they encrypt, or by any other specification. When a key group is created, you can associate it with a specific tape drive using the -symrec keyword in the **adddrive** command. See "adddrive" on page 92 for syntax.

In order to build a key group, you must define it in the KeyGroups.xml file. If you are creating the configuration file manually, the location of the KeyGroups.xml file is specified in the configuration properties file:

```
config.keygroup.xml.file = FILE:KeyGroups.xml
```

If this parameter is not specified, then the default behavior is to use the KeyGroups.xml file from the Security Key Lifecycle Manager for z/OS working directory of the launching location. If this file does not exist, an empty KeyGroups.xml file is created.

Key groups are built using the following CLI commands (see "Command Line Interface Commands" on page 92 for syntax):

Using CLI Commands to define key groups

The Security Key Lifecycle Manager for z/OS has a key group feature that allows you to group sets of keys.

When the Security Key Lifecycle Manager for z/OS application is installed and configured and the Security Key Lifecycle Manager for z/OS server is started, follow these steps:

1. Run the **createkeygroup** command.

   This command creates the initial key group object in the KeyGroups.xml file. Run this command once.

   Syntax: **createkeygroup -password** *password*

   -password
   The *password* that is used to encrypt the password of the keystore in the KeyGroups.xml file for later retrieval. The keystore encrypts the key of the key group, which in turn encrypts each individual key group alias password. Therefore no key in the KeyGroups.xml file is in the clear.

   Example: createkeygroup -password a75xynrd
2. Run the **addkeygroup** command.

   This command creates an instance of a key group with a unique group ID in the KeyGroups.xml.

   Syntax: **addkeygroup -groupID** *groupname*

-groupID
The unique *groupname* used to identify the group in the KeyGroups.xml file.

Example: addkeygroup -groupID keygroup1

3. Run the **addkeygroupalias** command.

   This command creates an alias for an existing key alias in your keystore for addition to a specific key group ID.

   Syntax: **addkeygroupalias -alias** *aliasname* **-groupID** *groupname*

   -alias
   The new *aliasname* for the key. This name must be the full key name. That is, Key00 must be entered as key000000000000000000.

   -groupID
   The unique *groupname* used to identify the group in the KeyGroups.xml file.

   Example: addkeygroupalias -alias key000000000000000000 -groupID keygroup1

   Note: When using this CLI command, you can only add one key at time. This command must be run for every individual key that must be added to the key group.

4. Associate a key group with a new or existing tape drive.
   a. Run the **moddrive** command to associate a key group with an existing tape drive.

      This command modifies tape drive information in the device table.

      Syntax: **moddrive -drivename** *drivename* **-symrec** *alias*

      -drivename
      drivename specifies the serial number of the tape drive.

      -symrec
      Specifies an *alias* (of the symmetric key) or a key group name for the tape drive.

      Example: moddrive -drivename 000123456789 -symrec keygroup1

   b. Run the **adddrive** command to add a tape drive to the device table and associate it with a key group.

      This command allows you to add a drive and associate it with a specific key group.

      Syntax: **adddrive -drivename** *drivename* **-symrec** *alias*

      -drivename
      drivename specifies the 12-digit serial number of the drive to be added.

      -symrec
      Specifies an *alias* (of the symmetric key) or a group ID for the tape drive.

      Example: adddrive -drivename 000123456789 -symrec keygroup1

When no alias is defined for a tape drive, specify a key group to be used as default for use. Set the symmetrickeySet property of the configuration properties file to the GroupID of the key group you want to use. For example,

```
symmetricKeySet = keygroup1
```

The group ID must match an existing key group ID in the KeyGroups.xml file. If not, the Security Key Lifecycle Manager for z/OS Server does not start. The

Security Key Lifecycle Manager for z/OS tracks key usage within a key group. When you specify a valid group ID, the Security Key Lifecycle Manager for z/OS records which key was last used. It then selects a random key from within the specified key group.

Copying keys from one key group to another

Run **addaliastogroup** command.

This command copies a specific alias from an existing (source) key group to a new (target) key group.

Syntax: **addaliastogroup -aliasID** *aliasname* **-sourcegroupID** *groupname* **-targetgroupID** *groupname*

-aliasID
The *aliasname* for the key to be added.

-sourcegroupID
The unique *groupname* used to identify the group from which the alias is to be copied.

-targetgroupID
The unique *groupname* used to identify the group to which the alias is to be added.

Example: addaliastogroup -aliasID aliasname -sourcegroupID keygroup1 -targetgroupID keygroup2

Note: Key is available in both key groups.

Chapter 4. Configuring the Security Key Lifecycle Manager for z/OS

The Configuring topics explain how to deploy and configure Security Key Lifecycle Manager for z/OS.

Configuration strategies

Some configuration settings in the ISKLMConfig.properties.zos file provide shortcuts that may have effects you should know about.

Automatically update device table

The Security Key Lifecycle Manager for z/OS provides variables in the configuration file to automatically update the device table. These variables are drive.acceptUnknownDrives and ds8k.acceptUnknownDrives (for DS8000). If you set the value to `true`, the device table is automatically populated when a new device contacts Security Key Lifecycle Manager for z/OS. This method eliminates the need to use the **adddrive** command for each tape drive or library. In this mode, the 12-digit serial number for each of these devices need not be entered using the CLI commands. The new drives undergo the normal public and private key cryptography exchange to verify the identity of the tape device. When this verification is complete, the new device is able to read existing tapes based on the EEDKs or key IDs stored on them. This is if the corresponding key information is found in the configured keystore.

Note: The Security Key Lifecycle Manager for z/OS server must be refreshed using the command "refresh" on page 97 after drives are added automatically. This ensures that they are stored in the device table.

For DS8000, the device is automatically associated with the certificates that are configured on the device when it makes a request to the Security Key Lifecycle Manager for z/OS.

For TS1120, TS1130, or TS1140 drives, this capability allows you to set the default certificate alias (or key label). This capability also allows you to set an alternate certificate alias (drive.default.alias1, drive.default.alias2) for encryption on newly added devices. For LTO Ultrium 4 and LTO Ultrium 5 drives, you can set the default symmetric key pool (symmetricKeySet) for encryption on newly added devices. You can have the Security Key Lifecycle Manager for z/OS fully configure the device with associated key material when the device makes contact. You can also configure the device after the tape drive has been added to the device table, using the **moddrive** command.

This feature relieves the administrator from entering the 12-digit serial number for each of the tape drives the Security Key Lifecycle Manager for z/OS will service. It also allows a default environment for large systems configurations.

The devices are added automatically and can be associated with a certificate alias (able to write a tape with that certificate alias). The added security check that the administrator would perform when adding the devices manually is skipped. As a consequence, there can be a slight reduction in security. It is important that you evaluate the advantages and disadvantages of this option. Determine if

automatically adding the tape drive information to the device table, and implicitly granting that new device access to the certificate information is an acceptable security risk.

Note: The drive.acceptUnknownDrives and ds8k.acceptUnknownDrives properties are set to `false` by default. Thus, the Security Key Lifecycle Manager for z/OS does not allow new drives to the device table automatically. Choose the mode you want to operate in and change the configuration accordingly.

Global default alias (key label) for TS1120, TS1130, and TS1140 tape drive writes

An option is available to set the Security Key Lifecycle Manager for z/OS default for the primary alias and secondary aliases. These aliases are used to wrap the data encrypting key. They are used when writing to a TS1120, TS1130, or TS1140 Tape Drive that is not supplied with key labels. The values in these two variables are used when a tape device in the device table does not have a defined alias to use when writing a tape. The Security Key Lifecycle Manager for z/OS then uses the drive.default.alias1 and drive.default.alias2 variables if they are set.

Note: The global default alias must be set in the ISKLMConfig.properties.zos file if:

- the tape drive is not supplied with key labels
- the acceptUnknownDrives setting is true
- a previously unknown tape drive communicates with the Security Key Lifecycle Manager for z/OS

Otherwise, the newly accepted drive cannot write tapes. The values in drive.default.alias1 and 2 must be defined with alias/key labels for this setting to work correctly. On the TS3500, the default alias (either defined in the properties file or the drive alias) is used to verify the Security Key Lifecycle Manager for z/OS operation. The test is done using a three-part test. If at least one set of aliases is not defined, the third part of the test fails, even though the Security Key Lifecycle Manager for z/OS is functional. See *IBM System Storage TS3500 Tape Library Operator Guide* for instructions on performing the test.

Synchronizing data between two Security Key Lifecycle Manager for z/OS servers

The device table and configuration properties file can be synchronized between two Security Key Lifecycle Manager for z/OS servers. Synchronization can be done manually by using the CLI client **sync** command. You can also synchronize automatically by setting four properties in the ISKLMConfig.properties.zos file.

Note: Neither synchronization method acts on the keystore or key groups XML file. They must be copied manually. If your environment uses Shared HFS function for Unix Systems Services and you use shared directories for debug and error logs, use of the **sync -all** or **sync -config** command is not commonly done. Using the commands changes the log settings on synchronized systems to use the same directories. Only the **sync -drivetable** command should be used for this type of configuration. The automatic synchronization function is only enabled when a valid IP address is specified in the sync.ipaddress property of the ISKLMConfig.properties.zos file. See “Automatic Synchronization” on page 81. Neither synchronization method acts on the keystore or key groups XML file. They must be copied manually.

Manual Synchronization

The manual method involves using the CLI client **sync** command. The syntax is as follows:

sync {**-all** | **-config** | **-drivetab**} **-ipaddr** *ip_addr :sslport* [**-merge** | **-rewrite**]

This command sends the configuration file properties or the device table information or both. The file or information is sent from the source server to the destination server specified by the **–ipaddr** parameter. The receiving Security Key Lifecycle Manager for z/OS server must be up and running.

Required fields

-all
Send both the configuration properties file and the device table information to the server specified by **-ipaddr**.

-config
Send only the configuration properties file to the server specified by **-ipaddr**.

-drivetab
Send only the device table information to the server specified by **-ipaddr**.

-ipaddr
ip_addr:sslport specifies the address and ssl port of the receiving server. The *sslport* must match the value specified for "TransportListener.ssl.port" in the ISKLMConfig.properties.zos file of the receiving server.

Optional fields

-merge
Merge (add) new device table data with current data on receiving server. (The configuration file is always a rewrite.) This setting is the default setting.

-rewrite
Replace the current data on the receiving server with new data.

Automatic Synchronization

The device table and properties file can be sent from a primary Security Key Lifecycle Manager for z/OS server to a secondary server automatically. The secondary server must be running for synchronization of the data to occur. To automatically synchronize the data from the primary to the secondary, the following four properties in the primary server ISKLMConfig.properties.zos file must be specified. There are no changes required to the secondary or receiving server properties file.

sync.ipaddress
Specifies the address and ssl port of the receiving server, for example,

```
sync.ipaddress = backupisklm.server.ibm.com:1443
```

If this property is unspecified or specified incorrectly, automatic synchronization is disabled.

sync.action
Merge or rewrite the existing data in the receiving server Valid values are **merge** (default) and **rewrite**. Synchronizing the configuration properties always results in a rewrite.

sync.timeinhours
How often the data must be sent. The value is specified in whole numbers (hours). The time interval begins when the server is started. The synchronization will occur after the server has been running for the specified number of hours. The default is 24 hours.

sync.type
Which data must be sent. Valid values are **drivetab** (default), **config**, and **all**.

If you are using hardware cryptography

Review the following to ensure that your environment meets all requirements.

1. Use one of the following keystore types:

   JCECCAKS
   JCECCARACFKS
   JCERACFKS

   Attention: You cannot use both JCERACFKS and JCECCARACFKS keystore types concurrently in the Security Key Lifecycle Manager for z/OS configuration file. You must specify only one of these types in the configuration file. If the JCERACFKS and JCECCARACFKS keystore types are used concurrently, the Security Key Lifecycle Manager for z/OS server will not start.
2. If you want the RSA key to be secure and not visible in the clear, create your RSA keys in the ICSF PKDS. Do this using either the RACDCERT PCICC option or **hwkeytool** with the **-hardwaretype PKDS** flag.
3. If you want the data encryption key to be secure and not visible in the clear, change the configuration to set the requireHardwareProtectionForSymmetricKeys property to true.
4. Ensure that the IBMJCECCA provider is installed in your java.security provider list.

If you are not using hardware cryptography

Ensure that your environment meets all the requirement requirements.

- Use a JCEKS keystore type.
- Ensure IBM SDK 5.0 SR5 or SDK 6.0 is installed. For z/OS, the minimum Java level is 5.0 SR5.
- Obtain a list of all the aliases (or key labels) for the RSA keys that you want to use. See your keystore documentation for more info on how to get this information.
- Obtain a list of all the type Drive Serial Numbers you need register. This is optional if you set `drive.acceptUnknownDrives = true` for automatic addition of tape drives to device table and `ds8k.acceptUnknownDrives=true` to automatically accept new DS8000 drives.
- Edit the ISKLMConfig.properties.zos file, as shown in "Configuration Basics," to customize the entries appropriate for your installation.

Configuration Basics

This procedure contains the minimum steps necessary to configure the Security Key Lifecycle Manager for z/OS.

1. Use a management tool specific to the keystore type you have chosen to create a keystore. See "Which Keystore is Right for You" on page 36. When creating

the keystore, note the path, file name, and the names given to the certificates and keys. This information is used in later steps.

2. Create a keystore if none exists. Add or import the certificates and keys used with your tape drives to this new keystore. See "Generating Keys and Aliases for Encryption on LTO Ultrium 4 and LTO Ultrium 5" on page 72. Take note of the names given to the certificates and keys. This information is used in later steps.
3. Edit the **ISKLMConfig.properties.zos** to update the following values.

   Note: The Security Key Lifecycle Manager for z/OS must not be running when you edit the **ISKLMConfig.properties.zos** file. If you have previously started the Security Key Lifecycle Manager for z/OS server, you must exit it or any changes you make is not saved.

   a. **Audit.handler.file.directory** – specify a location where audit logs are to be stored.
   b. **Audit.metadata.file.name** – specify a fully qualified path and file name for the metadata XML file.
   c. **config.drivetable.file.url** – specify a location for information about drives that are known to the Security Key Lifecycle Manager for z/OS. This file is not required before starting the server or CLI client. If it does not exist, it is created during shutdown of the Security Key Lifecycle Manager for z/OS server.
   d. **TransportListener.ssl.keystore.name** – specify the path and file name of the keystore created in step 1.
   e. **TransportListener.ssl.truststore.name** - specify the path and file name of the keystore created in step 1.
   f. **Admin.ssl.keystore.name** - specify the path and file name of the keystore created in step 1.
   g. **Admin.ssl.truststore.name** - specify the path and file name of the keystore created in step 1.
   h. **config.keystore.file** - specify the path and file name of the keystore created in step 1.
   i. **drive.acceptUnknownDrives** - specify `true` or `false`. A value of true allows new tape drives that contact the Security Key Lifecycle Manager for z/OS to be automatically added to the device table. The default is false.
   j. **ds8k.acceptUnknownDrives** - specify `true` or `false`. A value of true allows a new DS8000 that contacts the Security Key Lifecycle Manager for z/OS to be automatically added to the device table. The default is false.
4. The following optional password entries can be added or omitted. If these entries are not specified in **ISKLMConfig.properties.zos**, the Security Key Lifecycle Manager for z/OS prompts for the keystore password during the startup of the server.

   a. **Admin.ssl.keystore.password** - specify the password of the keystore created in step 1.
   b. **config.keystore.password** - specify the password of the keystore created in step 1.
   c. **TransportListener.ssl.keystore.password** - specify the password of the keystore created in step 1.

When added to the **ISKLMConfig.properties.zos** file, the Security Key Lifecycle Manager for z/OS obfuscates these passwords for additional security. Obfuscating the passwords ensures that they do not appear in the clear in the properties file.

5. Save the changes to **ISKLMConfig.properties.zos**.
6. Start the Security Key Lifecycle Manager for z/OS server:

   To start Security Key Lifecycle Manager for z/OS with JCERACFKS:

   ```
   java -Djava.protocol.handler.pkgs=com.ibm.crypto.provider
   com.ibm.ltklm.ISKLMServer ISKLMConfig.properties.zos
   ```

   To start Security Key Lifecycle Manager for z/OS with JCECCARACFKS:

   ```
   java -Djava.protocol.handler.pkgs=com.ibm.crypto.hdwrCCA.provider
   com.ibm.ltklm.ISKLMServer ISKLMConfig.properties.zos
   ```

   To start Security Key Lifecycle Manager for z/OS with JCEKS or JCECCAKS:

   ```
   java com.ibm.ltklm.ISKLMServer ISKLMConfig.properties.zos
   ```

7. Configure a drive by entering the following at the # prompt:

   ```
   adddrive -drivename drive_name -recl cert_name -rec2 cert_name
   ```

   For example:

   ```
   # adddrive -drivename 000001365054 -rec1 key1c1 -rec2 key1c2
   ```

   followed by

   ```
   # listdrives -drivename 000001365054
   ```

   returns

   ```
   Entry Key:  SerialNumber = 000001365054

   Entry Key:  AliasTwo = key1c2

   Entry Key:  AliasOne = key1c1
    Deleted : false
    Updated : true
    TimeStamp : Sun Jul 03 17:34:44 MST 2007
   ```

8. Enter the **listdrives** command at the # prompt to ensure that the drive was successfully added.

Configuration Properties

The Security Key Lifecycle Manager for z/OS can take advantage of the z/OS hardware cryptography provided by z/OS ICSF. The feature can be used to improve the security characteristics of the data encryption key generated by the Security Key Lifecycle Manager for z/OS. The following configuration properties, requireHardwareProtectionForSymmetricKeys and zOSCompatibility, can be considered when running the Security Key Lifecycle Manager for z/OS on the z/OS platform.

The **requireHardwareProtectionForSymmetricKeys** and **zOSCompatibility** configuration properties implement enhanced symmetric key handling in support of a Security Key Lifecycle Manager for z/OS. This ensures that tape data encryption keys can be generated, wrapped, and rewrapped. These actions are done under multiple RSA keys utilizing z/OS ICSF services and residing in hardware-protected locations. The Security Key Lifecycle Manager for z/OS can be configured with these properties. When configured, keys that are sent or received from the tape drive and used to encrypt data do not appear in an unencrypted form in z/OS host storage. z/OS ICSF services and zSeries® hardware

cryptography can be used to secure the RSA key management of symmetric keys. They are handled in a manner that would prevent these keys from appearing in an unencrypted form in host storage.

The requireHardwareProtectionForSymmetricKeys flag dictates that symmetric keys that are generated using ICSF must be protected by the ICSF Master Key. This way the symmetric key will never show up in the system memory in the clear. This flag only affects the TS1120, TS1130, TS1140 and DS8000 devices when using a JCECCAKS and JCERACFCCAKS keystore. LTO devices are not affected since the keys are pre-generated.

Note: If you use the requireHardwareProtectionForSymmetricKeys flag for generating keys for LTO drives, these keys cannot be exported using the keytool -exportseckey option. The tapes written with these keys can only be read by the Security Key Lifecycle Manager for z/OS that served the key for write operations or by an Security Key Lifecycle Manager for z/OS that shares the same keystore. For information about Integrated Cryptographic Services Facility and how to do exports of protected keys, see http://publib.boulder.ibm.com/infocenter/zos/v1r11/topic/com.ibm.zos.r11.csfb400/pt2a.htm#pt2a for the callable service called Data Key Export (CSNBDKX).

The zOSCompatibility flag is used to identify the crypto capabilities of the z/OS system being used. This flag is typically used when hardware cryptography is being used on z/OS, ICSF. At one point, ICSF did not support the AES algorithm that Security Key Lifecycle Manager for z/OS uses and this flag was a work around for that issue. However, ICSF does support AES now, so this flag does not need to be used anymore.

Note: If you need to have the zOSCompatibility flag turned on one system, make sure that you have it turned on all systems that are serving keys to the same devices.

Note: If this flag is turned on and the ICSF that you are currently using now supports AES, then you can turn this flag off. This will not affect previously encrypted cartridges. Any new cartridges will require use of an AES key. Therefore, the default keygroup for LTO devices must contain AES keys and not DESede keys.

The Security Key Lifecycle Manager for z/OS, when configured with the zOSCompatibility property set to true, uses the configured JCE cryptographic provider. This configuration causes a 168- bit DESede key to be generated in lieu of a 256 - bit AES key. This key, wrapped using an RSA key which is protected by hardware cryptographic services, is then provided to the tape drive device. The tape drive device continues to use 256 - bit AES-GCM encryption. This procedure is done using the 168- bit key. That key is used to build a 256 - bit AES key that is then used for data encryption and decryption performed within the device. When the Security Key Lifecycle Manager for z/OS is running and **requireHardwareProtectionForSymmetricKeys** is set to true, this key is always encrypted in z/OS host storage. The following tables provide additional information about these Security Key Lifecycle Manager for z/OS configuration properties.

requireHardwareProtectionForSymmetricKeys configuration property

Table 9. requireHardwareProtectionForSymmetricKeys property

Value	Applies to	Description and usage
true | false	Writing and reading tapes on the z/OS platform only when using a Security Key Lifecycle Manager for z/OS started with any of the jcecca provider-based keystores.	If `true`, the data encryption key used with the JCECCAKS keystore protected by z/OS cryptographic hardware. Data encryption key generated for encryption and decryption only appears in host storage. It appears in an encrypted form that is protected by a hardware resident master key. This option only affects z/OS JCECCA provider keystore types that are supported as stated in this publication. It has no affect on other keystore types.

Creating Security Key Lifecycle Manager for z/OS configuration file

Create the Security Key Lifecycle Manager for z/OS configuration file in /u/isklmsrv and customize accordingly for your installation

Audit.handler.file.directory
Modify this parameter to a location where you want the Security Key Lifecycle Manager for z/OS to store the audit logs.

Audit.metadata.file.name
Specify a file name for the metadata XML file.

config.drivetable.file.url
Specify a location for information about drives that are known to the Security Key Lifecycle Manager for z/OS. This file is not required to exist before starting the Security Key Lifecycle Manager for z/OS server or Security Key Lifecycle Manager for z/OS Admin console. If it does not exist, it is created during shutdown of the Security Key Lifecycle Manager for z/OS server of Security Key Lifecycle Manager for z/OS Admin Console.

Admin.ssl.keystore.name
Admin.ssl.truststore.name
config.keystore.file
TransportListener.ssl.keystore.name
TransportListener.ssl.truststore.name
Specify the path and file name of the keystore created previously.

requireHardwareProtectionForSymmetricKeys
This option allows users to define if the data encryption key used with the JCECCAKS, JCECCA, or JCECCARACFKS keystores are to be protected by z/OS cryptographic hardware. Keys generated and used by the Security Key Lifecycle Manager for z/OS only appear in host storage. They appear in an encrypted form that is protected by a hardware resident master key.

drive.acceptUnknownDrives
Specify true or false. A value of true allows new tape drives that contact the Security Key Lifecycle Manager for z/OS to be automatically added to the device table. The default is false. If you specify true for this value, set drive.default.alias1 and drive.default.alias2 to the certificate alias and key label that you previously created.

ds8k.acceptUnknownDrives
Specify true or false. A value of true allows a new DS8000 that contacts the Security Key Lifecycle Manager for z/OS to be automatically added to the device table. The default is false.

The following example illustrates a Security Key Lifecycle Manager for z/OS configuration file using the JCECCARACFKS customized for a z/OS system that is using shared HFS where systemname = JA0.

```
Admin.ssl.keystore.name = safkeyring://ISKLMSRV/KLMRing
Admin.ssl.truststore.name = safkeyring://ISKLMSRV/KLMRing
Audit.event.outcome = success,failure
Audit.event.outcome.do = success,failure
Audit.event.types = all
Audit.event.types.backup = data synchronization, runtime, configuration management,
  resource management
Audit.eventQueue.max = 0
Audit.handler.file.directory = /isklmlogs/JA0/audit
Audit.handler.file.name = kms_audit.log
Audit.handler.file.size = 10000
Audit.metadata.file.name = /keylifecyclemanager/metafile.xml
config.drivetable.file.url = FILE:/u/isklmsrv/JA0/filedrive.table
config.keystore.file = safkeyring://ISKLMSRV/ISKLMRing
config.keystore.password = password
config.keystore.provider = IBMJCECCA
config.keystore.type = JCECCARACFKS
debug = none
debug.output = simple_file
debug.output.file = /isklmlogs/JA0/debug
drive.acceptUnknownDrives = true
drive.default.alias1 = ISKLMServer
drive.default.alias2 = ISKLMServer
fips = Off
requireHardwareProtectionForSymmetricKeys = true
TransportListener.ssl.ciphersuites = JSSE_ALL
TransportListener.ssl.clientauthentication = 0
TransportListener.ssl.keystore.name = safkeyring://ISKLMSRV/ISKLMServer
TransportListener.ssl.keystore.password = password
TransportListener.ssl.keystore.type = JCECCARACFKS
TransportListener.ssl.port = 1443
TransportListener.ssl.protocols = SSL_TLS
TransportListener.ssl.truststore.name = safkeyring://ISKLMSRV/ISKLMServer
TransportListener.ssl.truststore.type = JCECCARACFKS
TransportListener.tcp.port = 3801
```

Configuring Security Key Lifecycle Manager for z/OS for LTO Ultrium 4 and LTO Ultrium 5 encryption

The management of encryption keys is expected to be performed using existing keystore management utilities. Another way to manage the encryption keys is using manual synchronization (extract/export, send, receive, import/insert) of the keys into the key groups and keystores used by the set of Security Key Lifecycle Manager for z/OS employed. Using this feature, the names (key IDs, key aliases/labels, key group IDs) of the symmetric keys is apparent to the Security Key Lifecycle Manager for z/OS administrators. The key IDs are not meant to be private or sensitive information.

The expected administrative steps are:

1. Create or import a certificate and private key for Security Key Lifecycle Manager for z/OS-to-Security Key Lifecycle Manager for z/OS communications. See the appropriate topic for your operating environment in Chapter 3, “Installing the Security Key Lifecycle Manager for z/OS and Keystores,” on page 43.
2. Generate a set of symmetric encryption keys. See “Generating Keys and Aliases for Encryption on LTO Ultrium 4 and LTO Ultrium 5” on page 72
3. Create key groups, and populate with key aliases. See “Creating and managing key groups” on page 76.
4. For each Security Key Lifecycle Manager for z/OS, store these keys into the keystore. This requirement is implicit in the invocation of KeyTool with the **–storetype** `jceks` and **–keystore** <*filename*> options. After a suitable number of stand-alone (-alias) or ranged (-aliasrange) invocations have been issued with KeyTool, the named keystore file from the command can be specified as a value for the `config.keystore.file` environment variable. The Security Key Lifecycle Manager for z/OS supports formats other than JCEKS as well, see “Which Keystore is Right for You” on page 36.

   Note: If the zOSCompatibility property is set to `true`, then **–keyAlg** must be set to `DESede` (3-DES) with an implicit effective bit length of 168. Do not specify this information. Otherwise, **-keyalg** must be `AES` and **-keysize** must be `256`.
5. For each Security Key Lifecycle Manager for z/OS, change the configuration. Set the key groups or key aliases or ranges to refer to the newly created keys. This setting is done using the configuration file property, `symmetricKeySet`. These aliases can be set up to match the aliases set up from KeyTool from steps 1 and 2. All ranged and stand-alone aliases can be specified at the same time, delimited by commas.
6. After all configurations have been updated, for each Security Key Lifecycle Manager for z/OS, restart the Security Key Lifecycle Manager for z/OS to incorporate the configuration changes.

z/OS Java Levels

Ensure that a level of Java for z/OS is installed that contains the JZOS launcher code. For SDK 5.0 this is SR 5 or higher. See http://www-03.ibm.com/servers/eserver/zseries/software/java for complete installation instructions on the Java for z/OS.

Note about z/OS configuration steps for z/OS in-band encrypted tape drive

If you are using z/OS in-band key management, there are additional configuration steps that you need to do. One of these steps is the identification of the port that the z/OS IOS subsystem (proxy support) is using to communicate with the Security Key Lifecycle Manager for z/OS. The specification of this port must be specified in the Security Key Lifecycle Manager for z/OS configuration and the appropriate parmlib member used by the z/OS IOS component. See *z/OS DFSMS Software Support for IBM System Storage TS1130 and TS1120 Tape Drives (3592)*, SC26-7514. for information on setting up the z/OS IOS subsystem and specifying an appropriate port ID.

Chapter 5. Administering the Security Key Lifecycle Manager for z/OS

The Administering topics explain how to perform administrative tasks.

Administering Security Key Lifecycle Manager for z/OS includes initial migration steps and solving startup problems. The command line interface enables you to perform administrative tasks such as adding a drive and importing a device table.

Migrating Encryption Key Manager to Security Key Lifecycle Manager for z/OS

You can migrate Encryption Key Manager versions 1.0, 2.0, and 2.1 to Security Key Lifecycle Manager for z/OS.

Before you begin

Before you migrate Encryption Key Manager to Security Key Lifecycle Manager for z/OS, take these steps:

- Use the **refresh** command to refresh Encryption Key Manager. For more information, see "refresh" on page 97.
- Use the **stopekm** command to stop Encryption Key Manager to ensure that there is no data loss. Encryption Key Manager cannot be active during migration.
- Install Security Key Lifecycle Manager for z/OS on the same computer as Encryption Key Manager. See, Chapter 3, "Installing the Security Key Lifecycle Manager for z/OS and Keystores," on page 43.
- Copy and store critical Encryption Key Manager files in a secure location that is not in the Encryption Key Manager directory structure. Use these files to restore Encryption Key Manager, if necessary.

  Copy and store the keystore, including all keys and certificates that the configuration file references. Also copy the configuration file, device table, and metadata file. Also copy the key groups file, if it exists.

  Also examine these properties in the Encryption Key Manager configuration file to determine the files that you copy:

  – config.keygroup.xml.file
  – config.drivetable.file.url
  – Admin.ssl.keystore.name
  – Admin.ssl.truststore.name
  – TransportListener.ssl.truststore.name
  – TransportListener.ssl.keystore.name
  – config.keystore.file
  – Audit.metadata.file.name

  Note: If the keystores in use are RACF-based (JCERACFKS or JCERACFCCAKS), then you do not have to back up any of the files referenced in the listed configuration parameters that have a keystore or truststore. You must still back up the config.keygroup.xml.file, config.drivetable.file.url, and Audit.metadata.file.name parameters.

Procedure

1. Copy the files that you backed up from the Encryption Key Manager to the directory in which you installed Security Key Lifecycle Manager for z/OS. That is, copy the configuration file, device table, keygroups file, metadata file, and keystore file or files, if applicable.
2. Update the Security Key Lifecycle Manager for z/OS configuration file to specify the new path in which you installed Security Key Lifecycle Manager for z/OS. Changing the path protects the Encryption Key Manager environment from being overwritten, in case you need to use Encryption Key Manager again.
3. Ensure that the Security Key Lifecycle Manager for z/OS configuration file, named **ISKLMConfig.properties.zos**, contains the following properties:

   Audit.metadata.file.name
   Specify the fully qualified path and file name for the XML file in which metadata is saved. For example:

   ```
   Audit.metadata.file.name = /u/isklmsrv/metafile.xml
   ```

   config.keystore.password
   Specify the keystore password. For example:

   ```
   config.keystore.password = ISKLMKeys.jck_password
   ```

   The password value is initially stored in plain text that is obfuscated when Security Key Lifecycle Manager for z/OS starts.

   TransportListener.ssl.keystore.password
   Specify the keystore password. For example:

   ```
   TransportListener.ssl.keystore.password = SSLKeystore.jck_password
   ```

   The password value is initially stored in plain text that is obfuscated when Security Key Lifecycle Manager for z/OS starts.

   Admin.ssl.keystore.password
   Specify the keystore password. For example:

   ```
   Admin.ssl.keystore.password = SSLKeystore.jck_password
   ```

   The password value is initially stored in plain text that is obfuscated when Security Key Lifecycle Manager for z/OS starts.

   Note: If you choose to automatically add a DS8000, you can add the property `ds8k.acceptUnknownDrives= true` **after** you complete the migration.
4. After all configurations have been updated, restart the Security Key Lifecycle Manager for z/OS to incorporate the configuration changes. See “Starting and Stopping Security Key Lifecycle Manager on z/OS ” on page 70. If the Encryption Key Manager used JZOS, Security Key Lifecycle Manager for z/OS can also use the JZOS launcher to restart. See, “Setting up and running Security Key Lifecycle Manager for z/OS in Production Mode” on page 67.
5. Check the audit log file to ensure that the migration was successful and that no errors are logged. For example:

   ```
   Runtime event:[
     timestamp=Sun Oct 24 10:33:28 CDT 2010
     ComponentId=[threadId=Thread[main,5,main]]
     event source=com.ibm.ltklm.ISKLMServer
     outcome=[result=successful]
     event type=SECURITY_RUNTIME
     resource=[name=ISKLMAdmin;type=application]
     action=runISKLMServer
     user=[name=ISKLMAdmin]
   ```

```
  ]
Resource management event:[
  timestamp=Sun Oct 24 10:33:29 CDT 2010
  ComponentId=[threadId=Thread[main,5,main]]
  event source=com.ibm.ltklm.keygroups.KeyGroupManager
  outcome=[result=successful]
  event type=SECURITY_MGMT_RESOURCE
  action=retrieve
  user=[name=KMSAdmin]
  resource=[name=SKLMKeys.jck;type=file]
  ]
Resource management event:[
  timestamp=Sun Oct 24 10:33:29 CDT 2010
  ComponentId=[threadId=Thread[main,5,main]]
  event source=com.ibm.ltklm.keystore.KeyStoreLoader
  outcome=[result=successful]
  event type=SECURITY_MGMT_RESOURCE
  action=retrieve
  user=[name=KMSAdmin]
  resource=[name=ISKLMKeys.jck;type=file]
  ]
Runtime event:[
  timestamp=Sun Oct 24 10:33:30 CDT 2010
  ComponentId=[threadId=Thread[main,5,main]]
  event source=com.ibm.ltklm.v
  outcome=[result=successful]
  event type=SECURITY_RUNTIME
  resource=[name=ISKLM server;type=application]
  action=start
  user=[name=ISKLMAdmin]
  ]
Runtime event:[
  timestamp=Sun Oct 24 10:33:54 CDT 2010
  ComponentId=[threadId=Thread[Thread-7,5,main]]
  event source=com.ibm.ltklm.ISKLMServer
  outcome=[result=successful]
  event type=SECURITY_RUNTIME
  resource=[name=ISKLM server;type=application]
  action=stop
  user=[name=ISKLMAdmin]
  ]
```

Note: The Encryption Key Manager only supported a flat file audit log. Security Key Lifecycle Manager for z/OS adds the ability to use SMF to log the audit records. For more information, see "Configuring the System Management Facilities Audit log" on page 121.

Solving Security Key Lifecycle Manager for z/OS Startup Problems

Resolve startup problems by checking the path values.

If Security Key Lifecycle Manager for z/OS fails to start, determine whether you correctly specified the path values for Encryption Key Manager. For additional troubleshooting steps, see "Debugging Security Key Lifecycle Manager for z/OS Server problems" on page 103.

You might revert to using Encryption Key Manager to serve keys until you resolve Security Key Lifecycle Manager for z/OS problems. Do not run both Encryption Key Manager and Security Key Lifecycle Manager for z/OS concurrently.

Command Line Interface Commands

The Security Key Lifecycle Manager for z/OS provides a command set. The command set can be used to interact with the Security Key Lifecycle Manager for z/OS server from a command-line interface client.

Note: When using either a JCERACFKS or JCECCARACFKS keystore, be aware that specifying a key alias in Java 6 is case-sensitive. When you specify a key alias in the CLI as a parameter value, ensure that you use the exact case match when using Java 6. If you do not specify the exact case match, the key cannot be retrieved from the keystore and it cannot be used during key serving. Specifying a key alias in Java 5 is not case-sensitive.

addaliastogroup

Copy a specific alias from an existing (source) key group to a new (target) key group. This is useful when you want to add an alias that exists in one key group to a different key group.

addaliastogroup -aliasID *aliasname* **-sourcegroupID** *groupname* **-targetgroupID** *groupname*

-aliasID
The *aliasname* for the key to be added.

-sourcegroupID
The unique *groupname* used to identify the group from which the alias is to be copied.

-targetgroupID
The unique *groupname* used to identify the group to which the alias is to be added.

Example: addaliastogroup -aliasID aliasname -sourcegroupID keygroup1 -targetgroupID keygroup2

adddrive

Add a new drive to the device table. See "Automatically update device table" on page 79 to learn how to add tape drives to the device table automatically. See "Encryption Keys and the TS1120, TS1130, TS1140 Tape Drives " on page 26, "Encryption Keys and the LTO Ultrium 4 Tape Drive and LTO Ultrium 5" on page 29, and "Encryption Keys and the DS8000 " on page 31 for information about alias requirements.

adddrive -drivename *drivename* [**-rec1** *alias*] [**-rec2** *alias*][**-symrec** *alias*]

-drivename
drivename specifies the 12-digit serial number of the drive to be added.

-rec1
Specifies the *alias* (or key label) of the certificate of the drive.

-rec2
Specifies a second *alias* (or key label) of the certificate of the drive.

-symrec
Specifies an *alias* (of the symmetric key) or a key group name for the tape drive.

Example: adddrive -drivename 000123456789 -rec1 alias1 -rec2 alias2

addkeygroup

Create an instance of a key group with a unique Group ID in the Key Group XML.

addkeygroup -groupID *groupname*

-groupID
The unique *groupname* used to identify the group in the KeyGroup XML file.

Example: addkeygroup -groupID keygroup1

addkeygroupalias

Create an alias for an existing key alias in your keystore for addition to a specific key group ID.

addkeygroupalias -alias *aliasname* **-groupID** *groupname*

-alias
The new *aliasname* for the key.

-groupID
The unique *groupname* used to identify the group in the KeyGroup XML file.

Example: addkeygroupalias -alias aliasname -groupID keygroup1

createkeygroup

Create the initial key group object in the KeyGroups.xml file. Run this command only once.

createkeygroup -password *password*

-password
The *password* that is used to encrypt the password of the keystore in the KeyGroups.xml file for later retrieval. The keystore encrypts the key of the key group, which in turn encrypts each individual key group alias password. Therefore no key in the KeyGroups.xml file is in the clear.

Example: createkeygroup -password password

deletedrive

Delete a drive from the device table. Equivalent commands are **rmdrive, deldrive** or **removedrive**.

deletedrive -drivename *drivename*

-drivename
drivename specifies the serial number of the drive to be deleted.

Example: deletedrive -drivename 000123456789

delgroupalias

Delete a key alias from a key group.

delgroupalias -groupID *groupname* **-alias** *aliasname*

-groupID
The unique *groupname* used to identify the group in the KeyGroups.xml file.

-alias
The *aliasname* for the key alias to be removed.

Example: delgroupalias -groupID keygroup1 -alias aliasname

delkeygroup

Delete an entire key group.

delkeygroup -groupID *groupname*

-groupID
The unique *groupname* used to identify the group in the KeyGroups.xml file.

Example: delkeygroup -groupID keygroup1

exit

Exit CLI client and stop Security Key Lifecycle Manager for z/OS server. Equivalent command is **quit**.

Example: exit

export

Export a device table or Security Key Lifecycle Manager for z/OS server configuration file to the specified URI.

export {**-drivetab**|**-config**} **-uri** *uriname*

-drivetab
Export the device table.

-config
Export the Security Key Lifecycle Manager for z/OS server configuration file.

-uri
uriname specifies the location where the file is to be written.

Example: export -drivetab -uri FILE:///keylifecyclemanager/data/export.table

help

Displays the command-line interface command names and syntax. Equivalent command is **?**.

Example: help

import

Import a device table or configuration file from a specified URI.

import {**-merge**|**-rewrite**} {**-drivetab**|**-config**} **-uri** *uriname*

-merge
Merge the new data with current data.

-rewrite
Replace the current data with new data.

-drivetab
Import the device table.

-config
Import the configuration file.

-uri
uriname specifies the location from which the new data is to be taken.

Example: import -merge -drivetab -uri FILE:///keylifecyclemanager/data/export.table

list

List certificates and keys contained in keystore named by config.keystore.file property.

list [**-cert** |**-key**|**-keysym**][**-alias** *alias* **-verbose** |**-v**]

-cert
List certificates in the specified keystore.

-key
List all keys in the specified keystore.

-keysym
List symmetric keys in the specified keystore.

-alias
alias specifies a specific certificate to list.

-verbose|-v
Displays more information about the certificate.

Examples:
`list -v` lists everything in the keystore.

`list -alias mycert -v` lists all available data for the mycert alias if it exists in the config.keystore.file keystore.

listcerts

List certificates contained in keystore named by config.keystore.file property.

listcerts [**-alias** *alias* **-verbose** |**-v**]

-alias
alias specifies a specific certificate to list.

-verbose|-v
Displays more information about the certificate.

Example: listcerts -alias alias1 -v

listconfig

Lists the Security Key Lifecycle Manager for z/OS server configuration properties in memory, reflecting the current contents of the ISKLMConfig.properties.zos file plus any updates made with the **modconfig** command.

Example: **listconfig**

listdrives

List drives in device table.

listdrives [**-drivename** *drivename*]

-drivename
drivename specifies the serial number of the tape drive to list.

-verbose|-v
Displays more information about the tape drive.

Example: listdrives -drivename 000123456789

modconfig

Modify a property in the Security Key Lifecycle Manager for z/OS server configuration properties file, ISKLMConfig.properties.zos. Equivalent command is **modifyconfig**.

modconfig {**-set** | **-unset**} **-property** *name* **-value** *value*

-set
Set the specified property to the specified value.

-unset
Remove the specified property.

-property
name specifies the name of the target property.

-value
value specifies the new value for the target property when **-set** is specified.

Example: modconfig -set -property sync.timeinhours -value 24

moddrive

Modify drive information in the device table. Equivalent command is **modifydrive**.

moddrive -drivename *drivename* {**-rec1** [*alias*] | **-rec2** [*alias*]| **-symrec** [*alias*]}

-drivename
drivename specifies the serial number of the tape drive.

-rec1
Specifies the *alias* (or key label) of the certificate of the drive.

-rec2
Specifies a second *alias* (or key label) of the certificate of the drive.

-symrec
Specifies an *alias* (of the symmetric key) or a key group name for the tape drive.

Example: moddrive -drivename 000123456789 -rec1 newalias1

refresh

Tells the Security Key Lifecycle Manager for z/OS to refresh the debug, audit, and device table values with the latest configuration parameters.

Example: refresh

refreshks

Refreshes the keystore. Use this command to reload the keystore specified in **config.keystore.file**. Use this command if it was modified while the Security Key Lifecycle Manager for z/OS server was running. Use this command only when needed as it can degrade performance.

Example: refreshks

runscript

Allows a script that contains Security Key Lifecycle Manager for z/OS commands to be run.

-filename
Specifies the *filename* of the script to be run. This can contain the path and the filename, as well.

Example: runscript -filename /u/user/script

status

Displays whether the server is started or stopped.

Example: status

stopisklm

Stops the Security Key Lifecycle Manager for z/OS server.

Example: stopiskm

sync

Synchronizes the configuration file properties, device table information, or both. It is synchronized on another Security Key Lifecycle Manager for z/OS server with those on the server issuing the command.

Note:

Neither synchronization method acts on the keystore or KeyGroups.xml file. These must be copied manually.

If your environment uses Shared HFS function for Unix Systems Services and you use shared directories for debug and error logs, use of the **sync -all** or **sync**

-config command is not recommended. This is because the command changes the log settings on synchronized systems to use the same directories. Only the **sync -drivetable** command should be used for this type of configuration.

sync {**-all** | **-config** | **-drivetab**} **-ipaddr** *ip_addr* :*ssl*:*port* [**-merge** | **-rewrite**]

-all
Send both the configuration properties file and the device table information to the Security Key Lifecycle Manager for z/OS server specified by **-ipaddr**.

-config
Send only the configuration properties file to the Security Key Lifecycle Manager for z/OS server specified by **-ipaddr**.

-drivetab
Send only the device table information to the Security Key Lifecycle Manager for z/OS server specified by **-ipaddr**.

-ipaddr
ip_addr:ssl:port specifies the address and ssl port of the receiving Security Key Lifecycle Manager for z/OS server. The *ssl:port* must match the value specified for "TransportListener.ssl.port" in the ISKLMConfig.properties.zos file of the receiving server.

-merge
Merge new device table data with current data. (The configuration file is always a rewrite.) This setting is the default setting.

-rewrite
Replace the current data with new data.

Example: sync -drivetab -ipaddr remoteisklm.ibm.com:443 -merge

version

Displays the version of the Security Key Lifecycle Manager for z/OS server.

Example: version

Chapter 6. Problem Determination

The Problem Determination topics explain how to address problems encountered in a Security Key Lifecycle Manager for z/OS environment.

You can enable debugging for an individual component, multiple components, or all components of the Security Key Lifecycle Manager for z/OS.

There are three places to look for errors:

The Security Key Lifecycle Manager for z/OS audit log
Most error messages appear in the audit log. The location and file name are set in the **Security Key Lifecycle Manager for z/OS ISKLMConfig.properties.zos** file in the Audit.handler.file.directory and Audit.handler.file.name properties.

Standard Error (stderr)
When running the Security Key Lifecycle Manager for z/OS as a started task using the Security Key Lifecycle Manager for z/OS console wrappers, examine the Execution LOG for errors. When running the Security Key Lifecycle Manager for z/OS in the foreground using USS/OMVS, these errors appear where you have directed STDERR.

The Security Key Lifecycle Manager for z/OS debug log
The location is set in the **Security Key Lifecycle Manager for z/OS ISKLMConfig.properties.zos file** debug.output.file property. The data written to the file is controlled by the debug property. For space reasons, it is best that you initially set the property to debug = none. If an error is encountered while Security Key Lifecycle Manager for z/OS is running you can turn debug on. You can turn debug on by submitting the **modconfig –set –property debug –value all** command. If you run into a problem and did not get any debug information from the Security Key Lifecycle Manager for z/OS audit log or Standard Error, set debug=all.

Note: The debug log should only be turned on at the direction of IBM service while debugging a specific problem and must only be turned on for a limited time. The debug log captures large amounts of data which might fill up the file system and cause an outage.

This is a list of errors and their possible causes that you might see when running the Security Key Lifecycle Manager for z/OS:

Error 1

```
com.ibm.keymanager.j [Caused by java.security.PrivilegedActionException:
java.io.IOException: The private key of ISLKMSERVE is not a software or
icsf key. Error creating key entry because private key is not available.]
```

Possible causes: If you are using a RACF keystore type (JCECCARACFKS or JCERACFKS):

- This error can occur if the user ID running the Security Key Lifecycle Manager for z/OS is not the owner of the KeyRing/Private key. RACF only allows a private key to be retrieved by its owner.
- This error can occur when starting the Security Key Lifecycle Manager for z/OS. This error occurs if your keyring has a public key that does not contain a

corresponding private key, such as a business partners key and that key was not connected as CERTAUTH (see directions in “Business Partner and Remote z/OS Systems ” on page 58).

Error 2

```
Runtime event:[
  timestamp=Wed Sep 06 13:30:54 EDT 2006
  event source=com.ibm.keymanager.g.fb
  outcome=[result=unsuccessful]
  event type=SECURITY_RUNTIME
  message= ***Error: Information not available for protected private keys..
     ErrorCode=0xEE0F
  resource=[name= Drive Serial Number: 000001350808 WWN: 500507630F04BC1B
     Key Alias/Label[0]: Tape_Sol_Tst_Shr_Pvt_1024_Lbl_02;type=file]
  action=stop
```

Possible cause: This error can occur if unrestricted policy files were not installed. Refer to “Copying the unrestricted policy files” on page 44. This error usually appears in the Security Key Lifecycle Manager for z/OS audit log.

Note: This error can also occur with an EE31 error code and the same text. It can be resolved by installing the unrestricted policy files.

Error 3

```
# java.lang.NoClassDefFoundError: javax/crypto/b
        at javax.crypto.Cipher.a(Unknown Source)
        at javax.crypto.Cipher.getInstance(Unknown Source)
        at com.ibm.keymanager.g.b.a(b.java:189)
        at com.ibm.keymanager.g.fb.a(fb.java:937)
        at com.ibm.keymanager.g.fb.run(fb.java:1277)
```

Possible cause: The wrong version, or a corrupt copy, of unrestricted policy files was installed. This error is sent to STDERR (your job execution log) and not the Security Key Lifecycle Manager for z/OS audit log.

Error 4

```
***Error: no such provider: IBMJCE4758. ErrorCode=0xEE0F
Runtime event:[
  timestamp=Mon Sep 18 22:43:26 EDT 2006
  event source=com.ibm.keymanager.logic.MessageProcessor
  outcome=[result=unsuccessful]
  event type=SECURITY_RUNTIME
  message= ***Error: no such provider: IBMJCE4758. ErrorCode=0xEE0F
  resource=[name= Drive Serial Number: 000001350699 WWN: 500507630F0C851C;type=file]
  action=stop
  ]
```

Possible cause: The Java hardware provider has not been added to the java.security provider list. This action must be done each time there is a new Java installation/upgrade if you are planning to use ICSF hardware keys. See “Add the Java Hardware Provider (Only if Using ICSF)” on page 45.

Error 5

```
java.security.PrivilegedActionException: java.io.IOException: R_datalib
(IRRSDL00) error: error while getting certificate or trust info (8, 8, 80)
```

Possible cause: Quotation marks surround the keyring name specified in the ISKLMConfig.properties.zos file (for example, `config.keystore.file = safkeyring:"//ISKLMSRV/ISKLMRing"`). Remove the quotation marks.

Error 6

```
java.security.PrivilegedActionException: java.io.IOException: Failed validating certificate paths
        at java.security.AccessController.doPrivileged1(Native Method)
        at java.security.AccessController.doPrivileged(AccessController.java:351)
        at com.ibm.keymanager.b.a.a(a.java:23)
```

```
at com.ibm.keymanager.b.a.a(a.java:148)
at com.ibm.keymanager.b.a.b(a.java:138)
at com.ibm.keymanager.i.a.a.h(a.java:711)
at com.ibm.keymanager.i.a.a.c(a.java:595)
at com.ibm.keymanager.KMSAdminCmd.main(KMSAdminCmd.java:2)
at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:85)
at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:58)
```

Possible cause: A CA Certificate is not connected to the KeyRing (note: At least one CERTAUTH cert is required, even if all certificates are self-signed). This message can only be displayed in the debug log and occur when attempting to start the Security Key Lifecycle Manager for z/OS Server.

Error 7

java.lang.NoClassDefFoundError

```
java.lang.NoClassDefFoundError: com/ibm/keymanager/logic/EncryptionCBDQuery
:at com.ibm.keymanager.logic.RequestEEDKs.createMsg(RequestEEDKs.java:48)
:at com.ibm.keymanager.logic.RequestEEDKs.<init>(RequestEEDKs.java:39)
:at com.ibm.keymanager.logic.MessageProcessor.ProcessMessage(MessageProcessor.java:351)
:at com.ibm.keymanager.logic.MessageProcessor.run(MessageProcessor.java(Compiled Code))
```

Possible cause: Java is not available. Possibly the file system where Java is installed has been dismounted. If using in-band key management, you can also see this IOS error:

```
IOS628E ENCRYPTION ON DEVICE E0A4 HAS FAILED DUE TO COMMUNICATION TIME OUT
IOS000I E0A4,D6,IOE,01,0E00,,**,A0209A,ITSXZ071 948
804C08C022402751 0301FF0000000000 0000000000000092 2004E82062612111
ENCRYPTION FAILURE
CU = 03 DRIVE = 000000 ISKLM = 000000
```

Error 8

```
JVMJZBL2999T JvmExitHook entered with exitCode=-3,
javaMainReturnedOrThrewExcep
JVMJZBL1043N The Java virtual machine completed with
System.exit(-3)
```

or

```
ISKLM server is now terminating abnormally with a return code of 4093.
```

Possible cause: The Java version (or just the Encryption Key Manager JAR version) was replaced such that the Encryption Key Manager was upgraded from a build earlier than 20070503 to a build equal to or later than 20070503. If that is the case, you must define the Audit.metadata.file.name property in the ISKLMConfig.properties.zos file. This is the name of the XML file where metadata is saved. This property is required to start versions of Encryption Key Manager with build date 20070503 (when metadata support was added) and later. See Chapter 8, “Using Metadata,” on page 127. Check your current Encryption Key Manager version Before you decide to upgrade to the latest Encryption Key Manager.

Error 9

```
Hardware error from call CSNDSYI
java.lang.IllegalArgumentException: System Error: Key
unwrapping is not supported in AMODE(64).. ErrorCode=0xEE31
resource=[name= Drive Serial Number: ds8k_device1 WWN:
57574E414D453030 Key Alias/Label[0]: cert1;type=file]
```

Possible cause: This error can occur when using Java 6.0 for 64-bit SDK with the ICSF level not updated to HCR7770 and the `requiredHardwareProtectionForSymmetricKeys` is set to `true`. This error can occur when a JCECCAKS keystore type is used.

This error can be resolved by updating your ICSF version to HCR7770.

ICSF Hardware Error

Under some circumstances, an error may occur when Security Key Lifecycle Manager for z/OS tries to access ICSF hardware crypto functionality. Security Key Lifecycle Manager for z/OS captures the hardware error message from ICSF, writes the message to audit log, and closes the socket connection with tape drive or storage device. The error code and return code in the error message are in decimal values. A sample message is:`Hardware error from call CSNBRNG returnCode 12reasonCode 0.`

For more information, see, http://publib.boulder.ibm.com/infocenter/zos/v1r10/index.jsp?topic=/com.ibm.zos.r10.csfb400/rcrcdes.htm. It documents the error code and return code in both hexadecimal and decimal forms.

Check these important files for Security Key Lifecycle Manager for z/OS server problems

When the Security Key Lifecycle Manager for z/OS fails to start check these files to determine the cause of the problem.

- **Debug log**

  If you set the debug.output.file, debug.output and debug properties to log activities, check this file to determine the cause of the problem. For more information see, Chapter 6, “Problem Determination,” on page 99.
- **Audit log**
  - Audit logs contain records that were logged as the Security Key Lifecycle Manager for z/OS is processing.
  - The location of this file is specified by two properties in **ISKLMConfig.properties.zos**, the Security Key Lifecycle Manager for z/OS Server configuration properties file:
    - Audit.handler.file.directory – specifies which directory the audit log must be located
    - Audit.handler.file.name – specifies the file name of the audit log.
  - For more information about Audit, see Chapter 7, “Audit Records,” on page 117.
- When you are running Security Key Lifecycle Manager for z/OS on JZOS, you can additionally check **STDERR** when Security Key Lifecycle Manager for z/OS fails to launch. When it is launched successfully, the output message goes to the system log. This is the default behavior, the log message does not show up in **STDOUT**. You can customize the default setting provided in sample **PROCLIB.ISKLM** to redirect the message to a different log file. The successful launch message shown at the console is similar to this:

  - 16.08.54 TVT4139 STC02013 BPXM023I (IBMUSER) Loaded drive key store successfully

  - 16.08.54 TVT4139 STC02013 BPXM023I (IBMUSER) Loading admin keystore...

  - 16.08.54 TVT4139 STC02013 BPXM023I (IBMUSER) Starting the Security Key Lifecycle Manager 1.1-20110126

  - 16.08.54 TVT4139 STC02013 BPXM023I (IBMUSER) Processing Arguments 00

  - 16.08.55 TVT4139 STC02013 BPXM023I (IBMUSER) Contact IBM support at 1-800-IBM-SERV (1-800-426-7378) or through your normal business channel.

  - 16.08.55 TVT4139 STC02013 BPXM023I (IBMUSER) Processing

- 16.08.55 TVT4139 STC02013 BPXM023I (IBMUSER) Server is started
- 16.08.55 TVT4139 STC02013 BPXM023I (IBMUSER) Server is running. TCP
- port: 3801, SSL port: 443

An example STDERR log:

When the Security Key Lifecycle Manager for z/OS is installed as JZOS and the keystore passwords in the ISKLMConfig.properties.zos file are 128 characters in length or greater, the Security Key Lifecycle Manager for z/OS will fail to start because it has no way to prompt for a password of acceptable length. The native Security Key Lifecycle Manager for z/OS logs will contain entries similar to the following:

```
com.ibm.ltklm.KeyManagerException: Default keystore failed to load
at com.ibm.ltklm.keygroups.KeyGroupManager.loadDefaultKeyStore(KeyGroupManager.
at com.ibm.ltklm.keygroups.KeyGroupManager.init(KeyGroupManager.java:202)
at com.ibm.ltklm.ISKLMServer.init(ISKLMServer.java:387)
at com.ibm.ltklm.ISKLMServer.<init>(ISKLMServer.java:214)
at com.ibm.ltklm.ISKLMServer.getInstance(ISKLMServer.java:222)
at com.ibm.ltklm.ISKLMServer.main(ISKLMServer.java:2502)
at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:79
at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl
at java.lang.reflect.Method.invoke(Method.java:618)
at com.ibm.jzosekm.ISKLMConsoleWrapper.invokeMain(ISKLMConsoleWrapper.java:297)
at com.ibm.jzosekm.ISKLMConsoleWrapper.main(ISKLMConsoleWrapper.java:50)
```

Viewing the STDOUT and STDERR logs

The STDOUT and STDERR logs are useful when Security Key Lifecycle Manager for z/OS running on JZOS fails to launch. These log files help to determine the cause of the problem.

About this task

Note: This topic applies only if Security Key Lifecycle Manager for z/OS running on JZOS fails to launch. If Security Key Lifecycle Manager for z/OS launches successfully, the STDOUT and STDERR logs are not created.

Procedure

1. In ISPF, select **Primary Option** > **S** (System Display and Search Facility).
2. Type **ST** (Status of Jobs) at **Command Input**.
3. Find the jobname you want to view by specifying **Find <jobname>**. A list of jobs is displayed.
4. Type **?** at the **NP** column beside the job that you want to view.
5. Press **Enter**.
6. Type **V** at the **NP** column beside the STDERR or STDOUT.
7. Press **Enter** to view the contents of the STDERR or STDOUT log.

Results

The contents of the STDERR or STDOUT log is displayed.

Debugging Security Key Lifecycle Manager for z/OS Server problems

Most problems concerning the Security Key Lifecycle Manager for z/OS involve configuration or starting the server.

If the Security Key Lifecycle Manager for z/OS fails to start, check for a firewall.

Either a software firewall or a hardware firewall can be blocking the Security Key Lifecycle Manager for z/OS from accessing the port.

Server is not started. File name for XML metadata file must be specified in the configuration file.

The Audit.metadata.file.name entry is missing from the configuration file.

To correct this problem, add the Audit.metadata.file.name property to the **ISKLMConfig.properties.zos** configuration file.

Failed to start ISKLM.Mykeys. The system cannot find the specified file.

1. This error message occurs when the keystore entries in **ISKLMConfig.properties.zos** do not point to an existing file.
2. To correct this problem, ensure the following entries in the **ISKLMConfig.properties.zos** file point to existing, valid keystore files:

   Admin.ssl.keystore.name

   TransportListener.ssl.truststore.name

   TransportListener.ssl.keystore.name

   Admin.ssl.truststore.name

Failed to start Security Key Lifecycle Manager for z/OS. File does not exist = safkeyring://xxx/yyy

The error can be caused by specifying the wrong provider in the IJO variable in the Security Key Lifecycle Manager for z/OS environment shell script.

For JCECCARACFKS keystores use:

```
-Djava.protocol.handler.pkgs=com.ibm.crypto.hdwrCCA.provider
```

and for JCERACFKS keystores use:

```
-Djava.protocol.handler.pkgs=com.ibm.crypto.provider
```

Failed to start Security Key Lifecycle Manager for z/OS. Keystore was tampered with, or password was incorrect.

1. This error occurs if one or more of these entries in the properties file has the wrong value:

   config.keystore.password (corresponds to config.keystore.file)

   admin.keystore.password (corresponds to admin.keystore.name)

   transportListener.keystore.password (corresponds to transportListener.keystore.name)
2. This error can also occur if the wrong password is entered at the password prompt on start-up of the server.
3. If none of the passwords are in the configuration, you are prompted up to three times. You are prompted if all three keystores entries in the properties file are unique. If all of the entries in the properties are the same, then you are prompted once.

Failed to start Security Key Lifecycle Manager for z/OS. Invalid keystore format.

1. This error can occur when the wrong keystore type is specified for one of the keystore entries in the properties file.
2. If all of the keystore entries in the properties file point to the same file, the Security Key Lifecycle Manager for z/OS uses the config.keystore.type value. This value is used as the keystore type for all keystores.
3. When there is no type entry in the properties file for a particular keystore, the Security Key Lifecycle Manager for z/OS assumes that the type is `jceks`.

Failed to start the server. Listener thread is not up and running.

This error can occur for a number of reasons:

1. The following two entries in the **ISKLMConfig.properties.zos** file point to the same port:

   TransportListener.ssl.port

   TransportListener.tcp.port

   Each of the transport listeners must be configured to listen on its own port.
2. Either of those entries is configured to a port that is already in use by another service running on the same machine as the Security Key Lifecycle Manager for z/OS server. Find ports that are not used by another service and use those to configure the Security Key Lifecycle Manager for z/OS server.
3. This error can occur if one or both of the ports are lower than 1024. This error occurs if the user starting the Security Key Lifecycle Manager for z/OS server is not root. Modify the transport listener entries in the **ISKLMConfig.properties.zos** to use ports above 1024.

Error: Unable to find Secretkey in the config keystore with alias:MyKey.

The symmetricKeySet entry in properties file contains a key alias that does not exist in the config.keystore.file

To correct this problem, modify the symmetricKeySet entry in the configuration file. Modify the file to only contain aliases that exist in the keystore file designated by the config.keystore.file entry in **ISKLMConfig.properties.zos** OR add the missing symmetric key to the keystore.

The symmetric key algorithm must be DESede if the zOSCompatibility flag in the configuration file is set to true.

The zOSCompatibility setting in the **ISKLMConfig.properties.zos** file is set to true with symmetricKeySet pointing to an alias of a key that is AES

To correct this problem, set the zOSCompatibility entry in the configuration file to `false` OR set the value of symmetricKeySet to specify DESede keys.

The symmetric key algorithm must be AES-256 if the zOSCompatibility flag in the configuration file is set to false.

The zOSCompatibility setting in the **ISKLMConfig.properties.zos** file is set to false with symmetricKeySet pointing to an alias of a key that is DESede

To correct this problem, set the zOSCompatibility entry in the configuration file to `true` OR set the value of symmetricKeySet to specify AES keys.

No symmetric keys in symmetricKeySet, LTO drives cannot be supported.

This message is an information message. The Security Key Lifecycle Manager for z/OS server starts, but LTO drives cannot be supported on this instance of Security Key Lifecycle Manager for z/OS. This is not a problem if there are no LTO drives configured to communicate with this Security Key Lifecycle Manager for z/OS.

Security Key Lifecycle Manager for z/OS - Reported Errors

This section defines error messages that are reported by the Security Key Lifecycle Manager for z/OS and returned in the drive sense data. They are typically called fault symptom codes or FSCs. The table includes the error number, a short description of the failure, and corrective actions.

Table 10. Errors that are reported by the Key Lifecycle Manager

Error Number	Description	Action
EE02	Encryption Read Message Failure: DriverErrorNotifyParameterError: "Bad ASC & ASCQ received. ASC & ASCQ does not match with either of Key Creation/Key Translation/Key Acquisition operation."	The tape drive asked for an unsupported action. Ensure that you are running the latest version of the Security Key Lifecycle Manager for z/OS. Check the versions of drive or proxy server firmware and update them to the latest release, if needed. Enable debug tracing on the server. Try to recreate the problem and gather debug logs. If the problem persists, contact IBM for support. See “Whom Do I Contact for IBM Support?” on page 109.
EE0F	Encryption logic error: Internal error: "Unexpected error....."	Ensure that you are running the latest version of the Security Key Lifecycle Manager for z/OS. Check the versions of drive or proxy server firmware and update them to the latest release, if needed. Enable debug tracing on the server. Try to recreate the problem and gather debug logs. If the problem persists, contact IBM for support. See “Whom Do I Contact for IBM Support?” on page 109.
	Error: Hardware error from call CSNDDSV returnCode 12 reasonCode 0.	If using hardware cryptography, ensure that ICSF is started.
EE23	Encryption Read Message Failure: Internal error: "Unexpected error....."	The message received from the drive or proxy server cannot be parsed because of general error. Ensure that you are running the latest version of the Security Key Lifecycle Manager for z/OS. Enable debug on the server. Try to recreate the problem and gather debug logs. If the problem persists, contact IBM for support. See “Whom Do I Contact for IBM Support?” on page 109.

Table 10. Errors that are reported by the Key Lifecycle Manager (continued)

Error Number	Description	Action
EE25	Encryption Configuration Problem: Errors that are related to the device table occurred.	Ensure that the config.drivetable.file.url is correct in the ISKLMConfig.properties.zos file, if that parameter is supplied. Run the `listdrives -drivename <drivename>` command on the Security Key Lifecycle Manager for z/OS server. This command verifies whether the drive is correctly configured. For example, the drive serial number, alias, and certificates are correct. Ensure that you are running the latest version of the Security Key Lifecycle Manager for z/OS. Check the versions of drive or proxy server firmware and update them to the latest release, if needed. Enable debug tracing and try the operation again. If the problem persists, contact IBM for support. See "Whom Do I Contact for IBM Support?" on page 109.
EE29	Encryption Read Message Failure: Invalid signature	The message received from the drive or proxy server does not match the signature on it. Ensure that you are running the latest version of the Security Key Lifecycle Manager for z/OS. Enable debug on the server. Try to recreate the problem and gather debug logs. If the problem persists, contact IBM for support. See "Whom Do I Contact for IBM Support?" on page 109.
EE2B	Encryption Read Message Failure: Internal error: "Either no signature in DSK or signature in DSK can not be verified."	Ensure that you are running the latest version of the Security Key Lifecycle Manager for z/OS. Check the versions of drive or proxy server firmware and update them to the latest release, if needed. Enable debug tracing on the server. Try to recreate the problem and gather debug logs. If the problem persists, contact IBM for support. See "Whom Do I Contact for IBM Support?" on page 109.
EE2C	Encryption Read Message Failure: QueryDSKParameterError: "Error parsing a QueryDSKMessage from a device. Unexpected dsk count or unexpected payload."	The tape drive asked the Security Key Lifecycle Manager for z/OS to do an unsupported function. Ensure that you are running the latest version of the Security Key Lifecycle Manager for z/OS. Check the versions of drive or proxy server firmware and update them to the latest release, if needed. Enable debug tracing on the server. Try to recreate the problem and gather debug logs. If the problem persists, contact IBM for support. See "Whom Do I Contact for IBM Support?" on page 109.

Table 10. Errors that are reported by the Key Lifecycle Manager (continued)

Error Number	Description	Action
EE2D	Encryption Read Message Failure: Invalid Message Type	The Security Key Lifecycle Manager for z/OS received a message out of sequence or received a message that it does not know how to handle. Ensure that you are running the latest version of the Security Key Lifecycle Manager for z/OS. Enable debug on the server. Try to recreate the problem and gather debug logs. If the problem persists, contact IBM for support. See "Whom Do I Contact for IBM Support?" on page 109.
EE2E	Encryption Read Message Failure: Internal error: Invalid signature type	The message received from the drive or proxy server does not have a valid signature type. Ensure that you are running the latest version of the Security Key Lifecycle Manager for z/OS. Enable debug on the server. Try to recreate the problem and gather debug logs. If the problem persists, contact IBM for support. See "Whom Do I Contact for IBM Support?" on page 109.
EE30	Prohibited request.	An unsupported operation has been requested for a tape drive. Enter the correct, supported command for the target tape drive.
EE31	Encryption Configuration Problem: Errors that are related to the keystore occurred.	Check the key labels that you are trying to use or configured for the defaults. You can list the certificates that are available to the Security Key Lifecycle Manager for z/OS by using the listcerts command. If you know that you are trying to use the defaults, then run the listdrives -drivename *drivename* command. Run the command on the Security Key Lifecycle Manager for z/OS server to verify whether the drive is correctly configured (for example, the drive serial number, and associated aliases/key labels are correct). If the drive in question has no aliases/key labels associated with it, then check the values of default.drive.alias1 and default.drive.alias2. If this does not help or the alias/key label exists, then collect debug logs and contact IBM for support. See "Whom Do I Contact for IBM Support?" on page 109.
EE32	Unable to locate the key requested on a key for read request made by an LTO device. The key requested does not exist in the config.keystore.file.	The probable cause is either that tape was encrypted using a different Security Key Lifecycle Manager for z/OS with different keys. Another cause might be that the key that was used to encrypt this tape has been renamed or deleted from the keystore. Issue `list -keysym` and ensure that the request alias is in the keystore.

Table 10. Errors that are reported by the Key Lifecycle Manager (continued)

Error Number	Description	Action
EEE1	Encryption logic error: Internal error: "Unexpected error: EK/EEDK flags conflict with subpage."	Ensure that you are running the latest version of the Security Key Lifecycle Manager for z/OS. Check the versions of drive or proxy server firmware and update them to the latest release, if needed. Enable debug on the server. Try to recreate the problem and gather debug logs. If the problem persists, contact IBM for support. See "Whom Do I Contact for IBM Support?."
EF01	Encryption Configuration Problem: "Drive not configured."	The drive that is trying to communicate with the Security Key Lifecycle Manager for z/OS is not present in the device table. Ensure that the config.drivetable.file.url is correct in the ISKLMConfig.properties.zos file, if that parameter is supplied. Run the `listdrives` command to check whether the drive is in the list. If not, configure the drive manually by using the `adddrive` command with the correct drive information or set the "drive.acceptUnknownDrives" or "ds8k.acceptUnknownDrives" property to `true` using the `modconfig` command. Enable debug tracing and retry the operation. If the problem persists, contact IBM for support. See "Whom Do I Contact for IBM Support?."
EDC5111I	In OMVS, the configuration file permission is set so that only the owner can read or write the configuration file.	If you log on and you are not the owner of the configuration file, you do not have permission to write to the configuration file. You might encounter an error similar to this: `- java.io.FileNotFoundException: /u/isklmsrv/JA0/ISKLMConfig.properties.zos.JCECCARACFKS (EDC5111I Permission denied.)` You might encounter this error when stopping the server, running the refresh operation, or changing passwords. For best practices, log on using the user ID with owner permissions.

Whom Do I Contact for IBM Support?

Find out how to contact IBM when you need support for the product.

The entitlement for software support varies depending on the operating system on which Security Key Lifecycle Manager for z/OS is running. Support also depends on whether the support requirement is defect-related or implementation-related.

Table 11. IBM Support Contacts

Type of Support	IBM Operating Systems:zOS	
Defect Support	Contact IBM Service with the name of the IBM operating system or identifier and customer number.	Contact IBM Service with the machine type/model and serial number of the IBM Tape Library.
Implementation Support [1]	Contact SupportLine IBM Service.	Contact SupportLine IBM Service.

Table 11. IBM Support Contacts (continued)

Type of Support	IBM Operating Systems:zOS
[1] An IBM Supportline contract offers the best Security Key Lifecycle Manager for z/OS implementation assistance. Some basic implementation assistance can be obtained by contacting IBM Service. Use the same machine type-model that would be used to report a defect. Should your customer require more extensive implementation assistance, billable onsite services are available from IGS and Lab Services. Contact IGS Inside Sales (888-426-4343 option 3) to obtain a Statement of Work (SOW).	

If there is a defect, IBM Service is always the first point of contact. The method to engage IBM Software Service varies depending on the operating system on which Security Key Lifecycle Manager for z/OS is being run.

For the following IBM operating systems: z/OS contact IBM Service (For US Customers call 800-IBM-SERV). Select the software option, then identify the operating system and the same customer number that was used to order the operating system.

Note: The relevant operating system here is the operating system on which Security Key Lifecycle Manager for z/OS is running. It is not the operating system that is generating the encrypted IOs.

Messages

Understand the messages displayed in the admin console and how to resolve them.

The following messages can be generated by the Security Key Lifecycle Manager for z/OS and displayed on the admin console.

Failed to Add Drive

Text

```
Failed to add drive. Device table entry already exists in the Table.
```

Explanation

The **adddrive** command failed because the drive is already configured with the Security Key Lifecycle Manager for z/OS and exists in the device table.

Operator Response

Run the **listdrives** command to see if the drive is already configured with Security Key Lifecycle Manager for z/OS. If the drive already exists, the drive configuration can be changed using **moddrive** command. Run **help** for more information.

Failed to Archive the Log File

Text

```
Failed to archive the log file.
```

Explanation

The log file cannot be renamed.

Operator Response

Check file permissions and space on that drive.

Failed to Delete the Drive Entry

Text

```
Failed to delete the drive entry.
```

Explanation

deldrive command failed to delete the drive entry from the device table.

Operator Response

Check the command syntax using **help** and make sure parameters supplied are correct. Make sure the drive is configured with the Security Key Lifecycle Manager for z/OS using **listdrives** command. Please check the audit logs for more information.

Failed to Import

Text

```
Failed to import. File <URL of file> does not exist.
```

Explanation

Device table or configuration files cannot be imported.

Operator Response

Make sure the specified URL exists and has read permissions. Check the command syntax using **help**. Make sure the parameters are correct and retry.

File Name Cannot be Null

Text

```
File name cannot be null.
```

Explanation

Audit file name is not supplied through configuration properties for the Security Key Lifecycle Manager for z/OS. This parameter is a required configuration parameter.

System Response

The program stops.

Operator Response

Check that the property `Audit.handler.file.name` is defined in the configuration properties file supplied to Security Key Lifecycle Manager for z/OS and try restarting it.

File Size Limit Cannot be a Negative Number

Text

```
File size limit cannot be a negative number.
```

Explanation

`Audit.handler.file.size` property value in the Security Key Lifecycle Manager for z/OS configuration file must be a positive number.

System Response

The Security Key Lifecycle Manager for z/OS does not start.

Operator Response

Please specify a valid number for `Audit.handler.file.size` and try restarting the Security Key Lifecycle Manager for z/OS.

No Data to be Synchronized

Text

```
No data can be found to be synchronized with "sync".
```

Explanation

The sync command cannot identify any data to be synchronized.

Operator Response

Check the configuration file supplied exists. Check if device table is correctly configured in the configuration file using *config.drivetable.file.url*. Check the syntax using **help** and retry the **sync** command.

Invalid Input

Text

```
Error: Invalid Input. Make sure the command entered is correct and retry.
```

Explanation

The particular command syntax might not be correct.

Operator Response

Make sure the command entered is correct. Check the command syntax using **help**. Make sure parameters supplied are correct and try again.

Invalid SSL Port Number in Configuration File

Text

```
Invalid SSL port number in configuration file for input string: <port
number>.
```

Explanation

SSL port number supplied in the configuration file is not a valid number.

System Response

The Security Key Lifecycle Manager for z/OS does not start.

Operator Response

Specify valid port number for the `TransportListener.ssl.port` property in the configuration file when starting the Security Key Lifecycle Manager for z/OS. Try to restart.

Invalid TCP Port Number in Configuration File

Text

```
Invalid TCP port number in configuration file. For input string: <value of
TCP port in config file>
```

Explanation

TCP port number supplied in the configuration file is not a valid number.

System Response

The Security Key Lifecycle Manager for z/OS does not start.

Operator Response

Specify valid port number for the `TransportListener.tcp.port` property in the configuration file when starting the Security Key Lifecycle Manager for z/OS. Try to restart. The default TCP port number is 3801.

Must specify SSL port number in configuration file

Text

```
Failed to start the server. com.ibm.ltklm.KeyManagerException:
com.ibm.ltklm.KeyManagerException: Must specify SSL port number in
configuration file.
```

Explanation

SSL port number is a required property to be configured in configuration properties file. It is used for communication between Security Key Lifecycle Manager for z/OS servers in a multi-server environment.

System Response

The Security Key Lifecycle Manager for z/OS does not start.

Operator Response

Specify valid port number for the `TransportListener.ssl.port` property and try to restart the Security Key Lifecycle Manager for z/OS.

Must Specify TCP Port Number in Configuration File

Text

```
Failed to start the server. com.ibm.ltklm.KeyManagerException:
com.ibm.ltklm.KeyManagerException: Must specify TCP port number in
configuration file.
```

Explanation

TCP port number is a required property to be configured in configuration properties file. It is used for communication between the drive and the Security Key Lifecycle Manager for z/OS.

System Response

The Security Key Lifecycle Manager for z/OS does not start.

Operator Response

Specify valid port number for the `TransportListener.tcp.port` property and try to restart the Security Key Lifecycle Manager for z/OS. The default TCP port number is 3801.

Server failed to start

Text

```
Failed to start the server.
```

Explanation

The Security Key Lifecycle Manager for z/OS server cannot start because of configuration problems.

Operator Response

Check the parameters in the configuration file supplied. Check the logs for more information.

Sync failed

Text

```
Sync failed: <error code>
```

Explanation

Sync operation to synchronize the data between two Security Key Lifecycle Manager for z/OS servers failed.

Operator Response

Make sure IP address specified for remote Security Key Lifecycle Manager for z/OS server is correct and that computer is accessible. Make sure that the configuration file exists and contains correct device table information. Check the **sync** command syntax using **help**. Check the logs for more information.

The specified audit log file is Read Only

Text

```
The audit log file can not be opened for writing.
```

Explanation

Audit log file in the Security Key Lifecycle Manager for z/OS configuration specified by the property `Audit.handler.file.name` cannot be opened for writing.

System Response

The Security Key Lifecycle Manager for z/OS does not start.

Operator Response

Check the permissions on the given audit file and directory and try restarting the Security Key Lifecycle Manager for z/OS.

Unable to load the Admin keystore

Text

```
Loading admin keystore... Invalid keystore format
```

Explanation

Admin keystore supplied to the Security Key Lifecycle Manager for z/OS cannot be loaded. Admin keystore is used between Security Key Lifecycle Manager for z/OS servers for server-side communication in multi-server environment.

System Response

The Security Key Lifecycle Manager for z/OS does not start.

Operator Response

Check the configuration file setup. Make sure the properties `admin.keystore.file`, `admin.keystore.provider` and `admin.keystore.type` in the Security Key Lifecycle Manager for z/OS configuration file are correct. Ensure that the keystore file exists and has read permission. Make sure the password supplied for admin keystore either through `admin.keystore.password` property or entered on the command line is correct. Try restarting the Security Key Lifecycle Manager for z/OS.

Unable to load the keystore

Text

```
com.ibm.ltklm.KeyManagerException: Default keystore failed to load.
```

Explanation

Keystore specified to the Security Key Lifecycle Manager for z/OS cannot be loaded.

System Response

The Security Key Lifecycle Manager for z/OS does not start.

Operator Response

Check the configuration file setup. Make sure the properties `config.keystore.file`, `config.keystore.provider` and `config.keystore.type` in the Security Key Lifecycle Manager for z/OS configuration file are correct and the keystore file exists and has read permission. Make sure the password supplied for the Security Key Lifecycle Manager for z/OS keystore either through `config.keystore.password` property or entered on the command line is correct. Try restarting.

Unable to load the transport keystore

Text

```
Loading transport keystore... Invalid keystore format.
```

Explanation

Transport keystore supplied to the Security Key Lifecycle Manager for z/OS cannot be loaded. Transport keystore is used between Security Key Lifecycle Manager for z/OS servers for client side communication in multi-server environment.

System Response

The Security Key Lifecycle Manager for z/OS does not start.

Operator Response

Check the configuration file setup. Make sure the properties `TransportListener.ssl.keystore.file`, `TransportListener.ssl.keystore.provider` and `TransportListener.ssl.keystore.type` in the Security Key Lifecycle Manager for z/OS configuration file are correct and the keystore file exists and has read permission. Make sure the password supplied for admin keystore either through `TransportListener.ssl.keystore.password` property or entered on the command line is correct. Try restarting Security Key Lifecycle Manager for z/OS.

Chapter 7. Audit Records

The Auditing topics explain how to audit events in a Security Key Lifecycle Manager for z/OS environment.

Note: The audit record formats described in this chapter are not considered to be programming interfaces. The format of these records can change from release to release. The format is documented in this chapter in case some parsing of the audit records is wanted.

Audit Overview

Security Key Lifecycle Manager for z/OS provides System Management Facilities support for audit records. System Management Facilities is a z/OS service aid that collects information from various z/OS subsystems. The default configuration on z/OS routes all audit records to System Management Facilities type 83 sub-type 6 records.

You can format Security Key Lifecycle Manager for z/OS audit data using the RACF SMF data unload utility. For information about how to run the RACF SMF Data unload utility, see the z/OS Security Server RACF Auditor's Guide.

The audit subsystem writes textual audit records. They are written to a set of sequential files as various auditable events occur during the processing of requests by Security Key Lifecycle Manager for z/OS. The audit subsystem writes to a file (directory and file name are configurable). The file size of these files is also configurable. As records are written to the file, and the size of the file reaches the configurable size. The file is then closed and renamed based on the current timestamp. Another file is opened and records are written to the newly created file. The overall log of audit records is separated into configurable sized files. Their names sequenced by the timestamp of when the size of the file exceeds the configurable size.

To keep the amount of information in the audit log from growing too large, consider creating a script or program. Create the script to monitor the set of files in the configured audit container. As files are closed and named based on the timestamp, the contents of the contents is copied. It is then appended to the specified long-term, continuous log location and then cleared. Be careful not to remove or alter the file which is having records written to it by the Security Key Lifecycle Manager for z/OS while running. This file does not have a timestamp in the file name.

Audit Configuration Parameters

The following parameters are used in the configuration file of Security Key Lifecycle Manager for z/OS. The parameters are used to control:

- which events are logged in the audit log
- where the audit log files are written to
- the maximum size of the audit log files

Audit.event.types

Syntax

Audit.event.types={*type*[*;type*]}

Usage

Used to specify which audit types should be sent to the audit log. Possible values for configuration parameter are:

all	All event types
authentication	Authentication events
data_synchronization	Events that occur during synchronization of information between Security Key Lifecycle Manager for z/OS servers
runtime	Events that occur as a part of processing operations and requests sent to the Security Key Lifecycle Manager for z/OS
configuration_management	Events that occur as configuration changes are made
resource_management	Events that occur as resource (tape drive) settings in the Security Key Lifecycle Manager for z/OS are changed

An example specification for this configuration value is:

```
Audit.event.types=all
```

Another example is:

```
Audit.event.types=authentication;runtime;resource_management
```

Audit.event.outcome

Syntax

Audit.event.outcome={*outcome*[*;outcome*]}

Usage

Used to indicate whether events occurring as a result of successful operations, unsuccessful operations, or both are to be audited. Specify **success** for events to be logged which occur as a result of successful operations. Specify **failure** for events to be logged which occur as a result of unsuccessful operations.

An example specification for this configuration value is:

```
Audit.event.outcome=failure
```

To activate both successful and unsuccessful cases:

```
Audit.event.outcome=success;failure
```

Audit.handler.class

Syntax

Audit.handler.class=*default_value*

Usage

Used to specify the class that will handle logging audit data. The Security Key Lifecycle Manager for z/OS installation program sets a default value.

Default

```
com.ibm.ltklm.audit.file.SimpleFileSecurityEventHandler
```

The Security Key Lifecycle Manager for z/OS installation program sets this value as the default on z/OS systems. The property and its value *are visible* in the ISKLMConfig.properties.zos file. For more information on changing the value of this property, refer to information on changing the default audit setting from SMF to file-based in the *IBM Tivoli Key Lifecycle Manager Installation and Configuration Guide*

Audit.eventQueue.max

Syntax

Audit.eventQueue.max=*number_events*

Usage

Used to set the maximum number of event objects to be held in the memory queue. This parameter is optional but recommended. The default is zero.

```
Audit.eventQueue.max=0
```

Audit.handler.file.directory

Syntax

Audit.handler.file.directory=*directoryName*

Usage

This parameter is used to indicate what directory the audit record files are to be written to. This parameter must specify a directory in the UNIX System Services file system. If the directory does not exist, the Security Key Lifecycle Manager for z/OS attempts to create the directory. If not successful, however, the Security Key Lifecycle Manager for z/OS does not start. It is best that the directory exists before running the Security Key Lifecycle Manager for z/OS. The user ID under which the Security Key Lifecycle Manager for z/OS runs must have write access to the directory specified.

To set the directory to **/var/isklm/isklm1/audit**:

```
Audit.handler.file.directory=/var/isklm/isklm1/audit
```

Audit.handler.file.size

Syntax

Audit.handler.file.size=*sizeInKiloBytes*

Usage

This parameter is used to indicate the size limit upon which an audit file is closed and a new audit file is then written to. The actual size of the audit file can exceed

this value by several bytes as the file is closed after the size limit has been exceeded.

To set the maximum file size to roughly 2 megabytes, enter:

```
Audit.handler.file.size=2000
```

Audit.handler.file.name

Syntax

Audit.handler.file.name=*fileName*

Usage

Use this parameter to specify the base file name within the specified audit directory to use as the base name in creating audit log files. This parameter must contain only the base file name and not the fully qualified path name. The full name of the audit log file has the value corresponding to the time upon which the file was written appended to this name.

Consider an example where the Audit.handler.file.name value is set to **isklm.log**. The full name of the file is similar to: `isklm.log.2315003554`. Use the appended string to determine the order that the audit log files were created. Higher number values indicate newer audit log files.

The following example sets the base name to **isklm.log**:

```
Audit.handler.file.name=isklm.log
```

Audit.handler.file.multithreads

Syntax

Audit.handler.file.multithreads={yes | true | no | false}

Usage

If specified as **true**, a separate thread is used to write the event data to the audit log. This setting allows the current thread of execution (operation) to continue without waiting for the write to the audit log to complete. Use of multiple threads is the default behavior.

An example setting the base name to **true** is:

```
Audit.handler.file.multithreads=true
```

Audit.handler.file.threadlifespan

Syntax

Audit.handler.file.threadlifespan=*timeInSeconds*

Usage

This parameter is used to specify the maximum time a thread is expected to require in order to write an audit log entry. This value is used during cleanup processing to allow threads to complete their work before interrupting them. If a background thread has not completed its work within the time allotted by the

threadlifespan parameter, then upon cleanup processing, the thread is interrupted.

To set the expected time a thread to write to the audit log as 10 seconds, specify:

```
Audit.handler.file.threadlifespan=10
```

Configuring the System Management Facilities Audit log

Before you begin

Configure Security Key Lifecycle Manager for z/OS to use the System Management Facilities (SMF) auditing log. Use the SMF audit log to record activities such as authentication events and data synchronization. This task requires:

- IBMSKLM.jar
- Your configured keystore file. For example: isklm.jck. For more information about keystores, see "Managing Keystores" on page 38.
- Your configured configuration file. For example: ISKLMConfig.properties.zos.
- CKLJSMF which is provided in the SCKLSAMP library. Use it to unload SMF type 83 audit records from the data set.
- Optional: Update the directories for the Keygroups.xml file and Device table.

Note: If the SMF auditing log is not configured but the auditing feature is enabled, a simple flat file is used for auditing.

Procedure

1. Run these RACF commands as root:

   ```
   RDEFINE FACILITY IRR.RAUDITX UACC(NONE)
   PERMIT IRR.RAUDITX CLASS(FACILITY) ID(<userid>) ACCESS(READ)
   SETROPTS RACLIST(FACILITY) REFRESH
   ```

2. Copy the CKLJSMF to a JCL library, for example to USER.PROCLIB(SMF2LOG).
3. Create a volume as stated in the JCL file from the Interactive System Productivity Facility (ISPF) menu.
   a. Select **Primary Option** > **Utilities** (option 3).
   b. Select **Data Set** (option 2) from **Utility Selection**.
   c. Specify the data set at **Other Partitioned, Sequential or VSAM Data Set**. For example: USER1.PRIV.SMFOUT
   d. Type **A** at **Option** to allocate the data set.
   e. In **Allocate New Data**, use the following options for the volume size:
      - Record format: **VB**
      - Record length: **12288**
      - Blocksize: **24576**
4. Create the file SMFPRM01 in USER.PARMLIB by copying the existing configuration of the PARMLIB dataset to the new configuration property.
   a. Select **Primary Option** > **Utilities** (option 3).
   b. Select **Move/Copy** (option 3) from **Utility Selection**.
   c. Specify the name of the data set at **From Other Partitioned, or Sequential Data Set**. For example: 'SYS1.PARMLIB(SMFPRM00)'
   d. Type **C** at **Option** to copy the data set.

e. Specify the name of the data set you want to copy at **Other Partitioned, Sequential or VSAM Data Set, or z/OS UNIX file**. For example: `'USER.PARMLIB(SMFPRM01)'`

5. Edit the system type to accommodate SMF Type 83.

```
ACTIVE
NOPROMPT
DSNAME(SYS1.MAN1,SYS1.MAN2,SYS1.MAN3)
REC(PERM)
MAXDORM(3000)
STATUS(010000)
JWT(2400)
SID(&SMFID.)
LISTDSN
MEMLIMIT(NOLIMIT)
SYS(TYPE(4,5,28,30,37,38,39,70,71,72,73,74,75,76,77,78,83))
SYS(NODETAIL)
SYS(NOINTERVAL)
SYS(EXITS(IEFACTRT,IEFUJI,IEFU83,IDFU84,IEFU85,IEFUTL,IEFU29,EIFUJV))
SUBSYS(STC,EXITS(IEFACTRT,IEFUJI,IEFU29,IEFU83,IEFU84,IEFU85)
```

6. Update the configuration file to use the SMF adapter in ISKLMConfig.properties.zos by entering this parameter:

```
Audit.handler.class=com.ibm.ltkm.audit.smf.SMFSecurityEventHandler
```

7. Set the SMF parameter to start logging SMF records using the configuration you created. Use the following commands:

z/OS console

```
SET SMF = 01
```

ISPF

```
ISPF -> s.log ->  /SET SMF=01
```

8. Launch Security Key Lifecycle Manager for z/OS.

z/OS console

Use the command: `S ISKLM`

Unix System Services (formerly known as OMVS)

Use the `java com.ibm.ltklm.ISKLMServer ISKLMConfig.properties.zos` command.

9. Run the **CKLJSMF** to unload the audit records in ISPF.
   a. Select **Primary Option** > **Library** (option 1).
   b. Specify the name of the file where the audit records are unloaded at **Other Partitioned, Sequential or VSAM Data Set, or z/OS UNIX file**. For example: `USER.PROCLIB(SMF2LOG)`
   c. Specify **submit** in the command field.
10. View the audit record to verify that the system logs activities.
    a. Select **Primary Option** > **Library** (option 1).
    b. Specify the name of the file where the audit records are unloaded at **Other Partitioned, Sequential or VSAM Data Set, or z/OS UNIX file**. For example: `'USER1.PRIV.SMFOUT'`

Results

You should see that the audit record logs records activities such as authentication events and data synchronization.

Audit Record Format

All audit records use a similar output format which is described here. All audit records contain some common information including timestamp and record type, along with information specific to the audit event which occurred. The general format for audit records is shown here:

```
AuditRecordType:[
  timestamp=timestamp
  Attribute Name=Attribute Value
  ...
  ]
```

Each record spans multiple lines in the file. The first line of the record beginning with the audit record type beginning at the first character on the line, followed by a colon (;) and an opening left bracket ([). Subsequent lines associated with the same audit record are indented two (2) spaces to assist in readability of the log records. The last line for a single audit record contains a closing right bracket (]) indented two (2) spaces. The number of lines for each audit record varies. The variation is based on the audit record type and the additional attribute information that is provided with the audit record.

The timestamp for the audit records is based on the system clock of the system on which the Security Key Lifecycle Manager for z/OS is running. If these records are correlated based on timestamp with events occurring on other systems, some type of time synchronization can be used. The time synchronization ensures that the clocks of the various systems in the environment are synchronized to an acceptable level of accuracy.

Audit points

The Security Key Lifecycle Manager for z/OS can write audit records, based on configuration, for many events that occur during the processing of requests. In this section, the set of events that can be audited is described along with the audit record configuration category, which must be enabled in order for these audit records to be written to the audit files (see Table 12).

Table 12. Audit record types that the Security Key Lifecycle Manager for z/OS writes to audit files

Audit Record Type	Audit Type	Description
Authentication	`authentication`	Used to log authentication events
Data Synchronization	`data_synchronization`	Used to log data synchronization processing
Runtime	`runtime`	Used to log various important processing events which occur within the Security Key Lifecycle Manager for z/OS server while handling requests
Resource Management	`resource_management`	Used to log changes to how resources are configured to the Security Key Lifecycle Manager for z/OS
Configuration Management	`configuration_management`	Used to log changes to the configuration of the Security Key Lifecycle Manager for z/OS server

Audit Record Attributes

The following lists show the attributes available to each of the audit record types.

Authentication event

The format for these records is:

```
Authentication event:[
  timestamp=timestamp
  event source=source
  outcome=outcome
  event type=SECURITY_AUTHN
  message=message
  authentication type=type
  users=users
  ]
```

The `message` value only appears if information for it is available.

Data synchronization event

The format for these records is:

```
Data synchronization event:
  timestamp=timestamp
  event source=source
  outcome=outcome
  event type=SECURITY_DATA_SYNC
  message=message
  action=action
  resource=resource
  user=user
  ]
```

The `message` and `user` values only appear if information for them is available.

Runtime event

The format for these records is:

```
Runtime event:
  timestamp=timestamp
  event source=source
  outcome=outcome
  event type=SECURITY_RUNTIME
  message=message
  resource=resource
  action=action
  user=user
  ]
```

The `message` and `user` values only appear if information for them is available.

Resource management event

The format for these records is:

```
Resource management event:
  timestamp=timestamp
  event source=source
  outcome=outcome
  event type=SECURITY_MGMT_RESOURCE
  message=message
```

```
  action=action
  user=user
  resource=resource
  ]
```

The `message` value only appears if information for it is available.

Configuration management event

The format for these records is:

```
Configuration management event:
  timestamp=timestamp
  event source=source
  outcome=outcome
  event type=SECURITY_MGMT_CONFIG
  message=message
  action=action
  command type=type
  user=user
  ]
```

The `message` value only appears if information for it is available.

Audited Events

Table 13 describes the events that cause audit records to be created. The table lists the audit record type that is logged when this event occurs.

Table 13. Audit record types by audited event

Audited Event	Audit Record Type
User successfully authenticated	authentication
User authentication failed	authentication
Data successfully sent to other Security Key Lifecycle Manager for z/OS	data_synchronization
Error sending data to other Security Key Lifecycle Manager for z/OS	data_synchronization
sync command processed	data_synchronization
Error processing sync command	data_synchronization
Command line processing started	runtime
exit command received	runtime
Unknown command entered	runtime
Message received from drive	runtime
Error processing message from drive	runtime
Error from message received from drive	runtime
Error updating device table with information received from drive	runtime
Error retrieving information from device table	runtime
Error retrieving information from keystore	runtime
Error processing certificate from keystore	runtime
Error finding private key from keystore	runtime
Error computing cryptographic values	runtime

Table 13. Audit record types by audited event (continued)

Audited Event	Audit Record Type
Message exchange processed successfully	runtime
Message processing started	runtime
Command line processing started	runtime
Problem found using cryptographic services	runtime
New drive discovered	runtime
Error configuring drive to device table	runtime
Successfully started processing messages from drive	runtime
Received and processed stopisklm command	runtime
Drive removed from device table	resource_management
Error removing drive from device table	resource_management
Device table import successful	resource_management
Error importing device table	resource_management
Device table export successful	resource_management
Error exporting device table	resource_management
listcerts command successful	resource_management
Drive add to device table successful	resource_management
Error adding drive to device table	resource_management
listdrives command successful	resource_management
Error processing listdrives command	resource_management
Device table modify successful	resource_management
Error modifying device table	resource_management
Successful KeyStore open	resource_management
Error opening KeyStore	resource_management
Configuration property changed	configuration_management
Error changing configuration property	configuration_management
Configuration property deleted	configuration_management
Error deleting configuration property	configuration_management
Configuration import successful	configuration_management
Error importing configuration	configuration_management
Configuration export successful	configuration_management
Error exporting configuration	configuration_management
listconfig command successful	configuration_management

Chapter 8. Using Metadata

The Security Key Lifecycle Manager for z/OS must be configured to create an XML file that captures vital information as data is being encrypted and written to tape. This file can be queried by volume serial number to display the alias or key label that was used on the volume. Conversely, the file can be queried by alias to display all volumes associated with that key label/alias.

Note: If you do not configure a metadata file, the Security Key Lifecycle Manager for z/OS does not start.

As encryption processing is performed, the Security Key Lifecycle Manager for z/OS collects the following data:

- Drive Serial Number
- Drive WorldWideName
- Creation Date
- Key Alias 1
- Key Alias 2
- DKi
- VolSer

When the collected data reaches a certain limit, it is written to an XML file. The default limit, which can be set in the Security Key Lifecycle Manager for z/OS properties file (ISKLMConfig.properties.zos), is 100 records. Once the file is written, it can be queried as long as the Security Key Lifecycle Manager for z/OS is running. To prevent the file from growing too large, it is automatically rolled over to a new file after a maximum file size is reached. The default maximum file size for rollover, which can also be set in the Security Key Lifecycle Manager for z/OS properties file, is 1 MB. Only a current and a previous file version is saved. The values to set in the Security Key Lifecycle Manager for z/OS configuration properties file are:

Audit.metadata.file.name
Name of XML file where metadata is saved. This configuration is required.

Audit.metadata.file.size
The maximum file size. This is specified in kilobytes, before rolling the file over from current to previous version. This configuration is optional. The default is 1024 (1MB).

Audit.metadata.file.cachecount
The number of records to be cached before writing the metadata file. This configuration is optional. The default is 100.

XML File Format

The file contains records in the following format.

```
<KeyUsageEvent>
 <DriveSSN>FVTDRIVE0000</driveSSN>       -Drive Serial Number
 <VolSer>TESTER</volSer>           -Volume Serial
 <DriveWWN>57574E414D453030</driveWWN>     -drive WWN
 <keyAlias2>cert2</keyAlias2>         -Key Alias1
```

```
 <keyAlias1>cert1</keyAlias1>         - keyAlias2
 <dateTime>Tue Feb 20 09:18:07 CST 2007</dateTime>   - creation date
</KeyUsageEvent>
```

Note: For LTO Ultrium 4 and LTO Ultrium 5 drives there is only <keyAlias1></keyAlias1> record. The DKi is not recorded.

Querying the metadata XML file

Use the ISKLMDataParser tool to query the metadata file. This tool parses the XML file using Document Object Model (DOM) techniques and cannot be run from the Security Key Lifecycle Manager for z/OS command-line interface. It is invoked as follows:

java com.ibm.ltklm.tools.ISKLMDataParser -filename *full_path_to_metadata_file* {**-volser** *volser* | **-keyalias** *alias*}

metadata_path
: This path is the same directory path specified for the metadata file in Audit.metadata.file.name in the **ISKLMConfig.properties.zos** file.

-filename
: *filename* is required and must be the name of the XML metadata file. The name usually matches the name specified in theAudit.metadata.file.name property in the **ISKLMConfig.properties.zos** file.

-volser
: The volume serial number of the tape cartridge you are searching for in the XML file. Either **-volser** or **-keyalias** must be specified.

-keyalias
: The key label or alias you are searching for in the XML file. Either **-volser** or **-keyalias** must be specified.

Example

The metadata file name property in **ISKLMConfig.properties.zos** is set to a value of `metadata`. The file is located in your local directory where the Security Key Lifecycle Manager for z/OS runs. The following command would filter (display) only the XML records related to volser `72448`:

```
<jvm_path>/bin/java com.ibm.ltklm.tools.ISKLMDataParser -filename metadata -volser 72448
```

The output would be formatted as follows:

Table 14. Metadata Query Output Format

keyalias1	keyalias2	volSer	dateTime	driveSSN	dki
cert1	cert2	72448	Wed Mar 14 10:31:32 CDT 2007	FVTDRIVE0004	

Recovering from a corrupted metadata file

The Security Key Lifecycle Manager for z/OS metadata file can become corrupted if the Security Key Lifecycle Manager for z/OS is improperly shutdown. It can also be corrupted if the system where the Security Key Lifecycle Manager for z/OS is running crashes. Improper editing or modification of the metadata file can also corrupt it. The corruption is unnoticed until the ISKLMDataParser parses the metadata file. The ISKLMDataParser can fail with an error like the following:

```
[Fatal Error] EKMData.xml:17:16: The end-tag for element type "KeyUsageEvent"
must end with a '>' delimiter.
org.xml.sax.SAXParseException: The end-tag for element type "KeyUsageEvent"
must end with a '>' delimiter.
at org.apache.xerces.parsers.DOMParser.parse(Unknown Source)
at org.apache.xerces.jaxp.DocumentBuilderImpl.parse(Unknown Source)
at javax.xml.parsers.DocumentBuilder.parse(Unknown Source)
at com.ibm.ltklm.tools.ISKLMDataParser.getData(ISKLMDataParser.java:80)
at com.ibm.ltklm.tools.ISKLMDataParser.viewData(ISKLMDataParser.java:143)
at com.ibm.ltklm.tools.ISKLMDataParser.main(ISKLMDataParser.java:337)
```

If this error occurs, it is due to a missing XML ending tag for an element. The Security Key Lifecycle Manager for z/OS metadata file can be recovered to allow the ISKLMDataParser to parse the file again.

1. Make a backup copy of the Security Key Lifecycle Manager for z/OS metadata file.
2. Edit the Security Key Lifecycle Manager for z/OS metadata file.
3. In XML, there should be an initial tag and a corresponding ending tag for each piece of data or event.
   - Some examples of an initial tag:
     - <KeyUsageEvent>
     - <driveSSN>
     - <keyAlias1>
   - Some examples of an ending tag:
     - </KeyUsageEvent>
     - </driveSSN>
     - </keyAlias1>
4. Scan the file and look for unmatched tags. The error message from the ISKLMDataParser lists which tag is missing its ending tag.
5. When an unmatched tag is found, temporarily delete the event or add the necessary tags to complete the event.
   - The following excerpt from a Security Key Lifecycle Manager for z/OS metadata file shows a first KeyUsageEvent that has no ending tag:

     ```
     <KeyUsageEvent>
     <driveSSN>001310000109</driveSSN>
     <volSer>      </volSer>
     <driveWWN>5005076312418B07</driveWWN>
     <keyAlias1>key000000000000000000F</keyAlias1>
     <dki>6B65790000000000000000F</dki>
     <dateTime>Thu Aug 30 09:50:53 MDT 2007</dateTime>
     <KeyUsageEvent>
     <driveSSN>001310000100</driveSSN>
     <volSer>      </volSer>
     <driveWWN>5005076312418ABB</driveWWN>
     <keyAlias1>key00000000000000000000</keyAlias1>
     <dki>6B6579000000000000000000</dki>
     <dateTime>Thu Sep 06 16:49:39 MDT 2007</dateTime>
     </KeyUsageEvent>
     ```

     Adding a </KeyUsageEvent> between the lines <dateTime>Thu Aug 30 09:50:53 MDT 2007</dateTime> and <KeyUsageEvent> would complete the first <KeyUsageEvent>.

Repairing the file corruption allows the ISKLMDataParser to successfully parse the data.

Appendix A. Sample Files

Sample files are provided to guide you in using Security Key Lifecycle Manager for z/OS.

Sample startup daemon script

Attention: It is impossible to overstate the importance of preserving your keystore data. You will not be able to decrypt your encrypted tapes if you do not have access to your keystore. Save your keystore and password information.

z/OS Platforms

Create or edit contents as shown in the example. The text in italics are for comment purposes only and do not have to be entered. Review the installation documentation for the z/OS Java product. The documentation contains additional guidelines, annotated samples, and step by step installation instructions for the JZOS launcher function.

```
//ISKLM PROC JAVACLS='com.ibm.jzosekm.ISKLMConsoleWrapper' ,
//   ARGS=,                              < Args to Java class
//   LIBRARY='SYS1.SIEALNKE',            < STEPLIB FOR JVMLDM module
//   VERSION='14',                       < JVMLDM version: 14, 50, 56
//   LOGLVL='+T',                        < Debug LVL: +I(info) +T(trc)
//   REGSIZE='0M',                       < EXECUTION REGION SIZE
//   LEPARM=''
//*********************************************************************
//*
//* Stored procedure for executing the JZOS Java Batch Launcher
//*  Specifically, to execute the Enterprise Key Manager under JZOS
//*
//*********************************************************************
//ISKLM  EXEC PGM=JVMLDM&VERSION,REGION=&REGSIZE,
//   PARM='&LEPARM/&LOGLVL &JAVACLS &ARGS'
//STEPLIB  DD DSN=&LIBRARY,DISP=SHR
//SYSPRINT DD SYSOUT=*          < System stdout
//SYSOUT   DD SYSOUT=*          < System stderr
//STDOUT   DD SYSOUT=*          < Java System.out
//STDERR   DD SYSOUT=*          < Java System.err
//CEEDUMP  DD SYSOUT=*
//ABNLIGNR DD DUMMY
//*********************************************************************
//*  The following member contains the JVM environment script
//*********************************************************************
//STDENV DD DSN=USER.PLX4.PROCLIB(ISKLM2ENV),DISP=SHR
//*
```

Sample server configuration properties files

The following is a sample properties file with all of the keystore entries pointing to the same software keystore:

```
Admin.ssl.keystore.name = /keylifecyclemanager/testkeys
Admin.ssl.keystore.type = jceks
Admin.ssl.truststore.name = /keylifecyclemanager/testkeys
Admin.ssl.truststore.type = jceks
Audit.event.outcome = success,failure
Audit.event.types = all
Audit.eventQueue.max = 0
```

```
Audit.handler.file.directory = /keylifecyclemanager/audit
Audit.handler.file.name = kms_audit.log
Audit.handler.file.size = 10000
Audit.metadata.file.name = /keylifecyclemanager/metafile.xml
config.drivetable.file.url = FILE:///keylifecyclemanager/drivetable
config.keystore.file = /keylifecyclemanager/testkeys
config.keystore.provider = IBMJCE
config.keystore.type = jceks
fips = Off
TransportListener.ssl.ciphersuites = JSSE_ALL
TransportListener.ssl.clientauthentication = 0
TransportListener.ssl.keystore.name = /keylifecyclemanager/testkeys
TransportListener.ssl.keystore.type = jceks
TransportListener.ssl.port = 443
TransportListener.ssl.protocols = SSL_TLS
TransportListener.ssl.truststore.name = /keylifecyclemanager/testkeys
TransportListener.ssl.truststore.type = jceks
TransportListener.tcp.port = 3801
```

This is a sample properties file with all of the keystore entries pointing to a different keystore. Entries in bold differ from the first sample properties file above.

```
Admin.ssl.keystore.name = /keylifecyclemanager/adminkeys.jceks
Admin.ssl.keystore.type = jceks
Admin.ssl.truststore.name = /keylifecyclemanager/admintrustkeys
Admin.ssl.truststore.type = jceks
Audit.event.outcome = success,failure
Audit.event.types = all
Audit.eventQueue.max = 0
Audit.handler.file.directory = /keylifecyclemanager/audit
Audit.handler.file.name = kms_audit.log
Audit.handler.file.size = 10000
Audit.metadata.file.name = /keylifecyclemanager/metafile.xml
config.drivetable.file.url = FILE:///keylifecyclemanager/drivetable
config.keystore.file = /keylifecyclemanager/drive.keys
config.keystore.provider = IBMJCE
config.keystore.type = jceks
fips = Off
TransportListener.ssl.ciphersuites = JSSE_ALL
TransportListener.ssl.clientauthentication = 0
TransportListener.ssl.keystore.name = /keylifecyclemanager/sslkeys
TransportListener.ssl.keystore.type = jceks
TransportListener.ssl.port = 443
TransportListener.ssl.protocols = SSL_TLS
TransportListener.ssl.truststore.name = /keylifecyclemanager/ssltrustkeys
TransportListener.ssl.truststore.type = jceks
TransportListener.tcp.port = 3801
```

Appendix B. Configuration Properties Files

The Security Key Lifecycle Manager for z/OS requires a configuration file. This file is treated and parsed as a Java.util.Properties load file, which imposes certain restrictions on the format and specification of properties:

- Configuration properties are recorded one-per-line. The value(s) for a given property extend to the end of the line.
- Property values, such as passwords, that contain spaces need not be enclosed in quotation marks.
- Keystore passwords must not be greater than 127 characters in length.
- Accidental whitespace at the end of a line may be interpreted as part of a property value.

Server configuration properties file

This section describes the properties in the Security Key Lifecycle Manager for z/OS configuration file. The order of property settings in the file does not matter. Comments can be used in the file. To add a comment, use a "#" in the first column of a line. On z/OS, "#" is expected to be in IBM-1047 code page, for example X'7B'.

Note: Changes made to the ISKLMConfig.properties.zos file can be lost at shutdown. Ensure that the Security Key Lifecycle Manager for z/OS server is not running before editing configuration properties. To stop the Security Key Lifecycle Manager for z/OS server issue the **stopisklm** command from the console where Security Key Lifecycle Manager for z/OS was started. If you are using JZOS use the **S ISKLM** command. Your changes are activated when the Security Key Lifecycle Manager for z/OS server is restarted.

Admin.ssl.ciphersuites = *value*

Specifies the cipher suites to be used for communication between the Security Key Lifecycle Manager for z/OS servers. A cipher suite describes the cryptographic algorithms and handshake protocols Transport Layer Security (TLS) and Secure Sockets Layer (SSL) use for data transfer.

Required

Optional.

Values

Possible values are any cipher suites supported by IBMJSSE2.

Default

JSSE_ALL

Admin.ssl.keystore.name = *value*

This is the name of the database of key pairs and certificates used for Secure Socket Layer client operations. They are used in operations such as **sync** commands between the Security Key Lifecycle Manager for z/OS Servers. In a sync operation, the certificate that the Secure Sockets client presents to the Secure Sockets server comes from this keystore.

Required

Optional. Used only with **sync** command. Defaults to value of **config.keystore.file** property.

Admin.ssl.keystore.password = password
Password to access Admin.ssl.keystore.name

Required
Optional. If not supplied, can be prompted for on start of the Security Key Lifecycle Manager for z/OS. The value for this property is obfuscated for additional security and the stanza name itself in the properties file is replaced with a new stanza that is named Admin.ssl.keystore.password.obfuscated.

Admin.ssl.keystore.type = *value*
Type of keystore used.

Required
Optional.

Default
jceks

Admin.ssl.protocols = *value*
Security protocols.

Required
Optional.

Values
SSL_TLS | SSL | TLS

Default
SSL_TLS

Admin.ssl.timeout = *value*
Specifies how long a socket waits for a read() before throwing a SocketTimeoutException.

Required
Optional.

Values
Specified in minutes. 0 means no timeout

Default
1

Admin.ssl.truststore.name = *value*
This is the name of the database file used to check the trust of the Secure Sockets Server certificate that the server presents to the Secure Sockets client.

Required
Optional. Used only with **sync** command. Defaults to value of **config.keystore.file** property.

Admin.ssl.truststore.type = *value*
Type of keystore used.

Required
Optional.

Default
jceks

Audit.event.outcome = *value*
Only audit events that resulted in the specified outcome are recorded

Required
Yes.

Values
success | failure. Both can be specified separated by comma or semicolon.

Default
success

Audit.handler.class = *value*
Specifies the class used to log audit data.

Required
Optional.

Default
com.ibm.ltklm.audit.file.SimpleFileSecurityEventHandler

Audit.event.Queue.max = 0
The maximum number of event objects in the audit memory queue before they are flushed to file.

Required
Optional.

Values
0 - ? (0 means flush immediately.)

Default
0

Audit.event.types = *value*
Only audit events that resulted in the specified outcome are recorded

Required
Yes.

Values
all |data synchronization | runtime | configuration management | resource management. Multiple values can be specified separated by a comma or semicolon.

Default
all

Audit.handler.file.directory = ../audit
Directory where Audit.handler.file.name is located

Required
Optional.

Audit.handler.file.multithreads = *value*
Specifies if the audit handler dispatches separate threads to process audit records.

Required
Optional.

Values
true | false

Default
true

Audit.handler.file.name = kms_audit.log
File name where audit entries are set to be logged.

Required
Yes.

Audit.handler.file.size = 100
Size to which Audit.Handler.file.name grows to before it begins to overwrite

Required
Optional.

Values
0 - ? (specified in kilobytes.)

Default
100

Audit.handler.file.threadlifespan = *value*
Limits the lifetime of an audit record processing thread. Only useful if audit.handler.file.multithreads= true.

Required
Optional.

Values
Specified in milliseconds.

Default
10000

Audit.metadata.file.cachecount = 100
Specifies the number of records to store in memory before writing the metadata file.

Required
Optional.

Default
100

Audit.metadata.file.name = *value*
Specifies the name of the XML file where metadata records are to be saved.

Required
Yes.

Audit.metadata.file.size = 1024
Specifies the maximum file size, specified in KB, the XML metadata file can achieve before the file is closed and a new file is started. Only a current and previous version of the file is saved.

Required
Optional.

Default
1024

cert.valiDATE = *value*
Specifies whether to perform certificate date validation. A value of **true** specifies that the dates of notBefore and notAfter certificates are used to validate the certificate. Any certificate with a date range falling outside these Java date specifications cannot be used for encryption. They can however still be used to decrypt, or read, encrypted tapes. If this property

is set to a value of **false** (or any other value, or left unspecified), no other certificate date validation is performed. This property only applies to TS1120, TS1130, TS1140 and DS8000. It is not used for LTO.

Required
Optional.

Values
true | false

Default
false

config.drivetable.file.url = FILE:../filedrive.table
File containing information concerning the tape drive such as serial number, and certificates.

Required
Yes.

config.keygroup.xml.file = *value*
Specifies the name of the XML file where individual aliases are stored by key groups. (Not used for TS1120, TS1130, TS1140, or DS8000.)

Required
Optional.

config.keystore.file = *value*
Specifies the keystore to be used.

Required
Yes.

config.keystore.password = password
Password to access config.keystore.file. The value for this property is obfuscated for additional security and the stanza name itself in the properties file is replaced with a new stanza that is named config.keystore.password.obfuscated.

Required
Optional. If not supplied, can be prompted for on start of the Security Key Lifecycle Manager for z/OS.

config.keystore.provider = IBMJCE

Required
Optional.

config.keystore.type = jceks

Required
Optional.

Default
jceks

debug = *value*
Enables debug for the specified Security Key Lifecycle Manager for z/OS component.

Note: The debug log should only be turned on at the direction of IBM service while debugging a specific problem and must only be turned on for a limited time. The debug log captures large amounts of data which might fill up the file system and cause an outage.

Required
Optional.

Values
all | audit | server | drivetable | config | admin | transport | logic | keystore | console | none. Can take multiple values separated by commas.

Default
none

debug.output = *value*
Routes debug output to specified location.

Required
Optional.

Values
simple_file | console.

debug.output.file = debug
Path and filename where debug output is to be written.

Required
Optional. Required when debug.output = simple_file. Path to file must exist.

drive.acceptUnknownDrives = *value*
Automatically adds new drive contacting the Security Key Lifecycle Manager for z/OS to device table.

Required
Optional.

Values
true | false

Default
false

Note: When used with a valid drive.default.alias1 and drive.default.alias2 setting for TS1120, TS1130 and TS1140 devices, or used with a valid symmetricKeySet for LTO devices, this setting allows tape drives that connect to the Security Key Lifecycle Manager for z/OS to be added and operational without an administrator intervention.

ds8k.acceptUnknownDrives=*value*
Automatically adds new DS8000 contacting the Security Key Lifecycle Manager for z/OS to device table.

Required
Optional.

Values
true | false

Default
false

drive.default.alias1 = *value*
Specifies a TS1120, TS1130, or TS1140 drive default alias if one is not specified in device table. This value can be different from the value specified for drive.default.alias2 or the same. (Not used for LTO and DS8000.)

Required
Optional.

drive.default.alias2 = *value*
Specifies a TS1120, TS1130, or TS1140 drive default alias if one is not specified in device table. This value can be different from the value specified for drive.default.alias1 or the same. (Not used for LTO and DS8000.)

Required
Optional.

fips = *value*
Federal Information Processing Standard.

Required
Optional.

Values
on | off

Default
off

Note: Do not use hardware-based keystore types when the fips parameter is set on.

maximum.threads = 200
Maximum number of threads the Security Key Lifecycle Manager for z/OS can create.

Required
Optional.

requireHardwareProtectionForSymmetricKeys = *value*
Specifies whether symmetric keys need to be protected on z/OS if hardware crypto is used.

Required
Optional.

Values
true | false

Default
false

symmetricKeySet = {GroupID | keyAliasList [, keyAliasList,]}
Specifies the symmetric key aliases and key groups to be used for LTO Ultrium 4 and LTO Ultrium 5 tape drives. (Not used for TS1120, TS1130, TS1140 or DS8000.)

Required
Optional. Applies to LTO Ultrium 4 and LTO Ultrium 5 tape cartridges only.

Values

Specify one value for *GroupID* or one or more values for *keyAliasList*.

GroupID specifies a key group name to prime the list of symmetric keys and serve as default when no alias is specified for the tape drive. The *GroupID* must match an existing key group ID in the

KeyGroups.xml file. If not, a KeyManageException is returned. If more than one *GroupID* is specified, a KeyManagerException is returned. When you specify a valid *GroupID*, the last key used in the Key Groups XML is tracked and a random selection for the next key is used each time getKey is called from the KeyGroups.xml for the list of symmetric keys. Each specification of *keyAliasList* contains either a value for *keyAlias* or *keyAliasRange*.

keyAlias specifies the Backus-Naur Form (BNF) for a name or alias of a symmetric key in the keystore up to 12 characters long or a sequentialKeyID exactly 21 characters long.

keyAliasRange specifies a sequentialKeyID and hexadecimal digits up to 18 characters, separated by a hyphen (-). If 18 characters are specified the first two characters must be 00. Must be specified on one line and contain no cr-lf.

GroupID specifies the name of a group of aliases.

Example

`symmetricKeySet = KMA0238ab34,KMB0000034acd2345678a,THZ001-FF` This instructs the Security Key Lifecycle Manager for z/OS to use aliases KMA0238ab34, KMB0000034acd2345678a. It uses aliases range of THZ000000000000000001 through THZ000000000000000FF when serving keys to LTO Ultrium 4 and LTO Ultrium 5 tape drives. These keys must exist in the keystore specified by **config.keystore.file** in the properties file.

sync.action = *value*

Specifies what action to be taken with the data during an auto synchronize.

Required

Optional.

Values

rewrite | merge

Default

merge

Note: merging configuration information is the same as rewriting it.

sync.ipaddress = *ip_addr:sslport*

Specifies the IP address and port of the remote Security Key Lifecycle Manager for z/OS to auto synchronize.

Required

Optional. If this property is unspecified or specified incorrectly, the sync function is disabled.

Values

IP address of remote server:SSL port number

sync.timeinhours = *value*

Specifies how many hours to wait before doing an auto synchronize with a remote Security Key Lifecycle Manager for z/OS.

Required

Optional.

Values

Specified in hours.

Default
24

sync.type = *value*
Specifies what data to auto synchronize.

Required
Optional.

Values
config | drivetab | all

Default
drivetab

TransportListener.ssl.ciphersuites = JSSE_ALL
Cipher suites to be used for communication between Security Key Lifecycle Manager for z/OS servers. A cipher suite describes the cryptographic algorithms and handshake protocols Transport Layer Security (TLS) and Secure Sockets Layer (SSL) use for data transfer.

Required
Optional.

Values
Values – any cipher suites supported by IBMJSSE2.

TransportListener.ssl.clientauthentication = 0
SSL authentication needed for communication between Security Key Lifecycle Manager for z/OS servers.

Required
Optional.

Values

0 - no client authentication (default)
1 - server wants to do client authentication with the client
2 - the server must do client authentication with the client

TransportListener.ssl.keystore.name = *value*
The name of the database used by the Security Key Lifecycle Manager for z/OS Server to hold the certificate and private keys for the Secure Socket Server. This certificate is given to the Secure Socket client for authentication and trust checking. This keystore is also used by the Security Key Lifecycle Manager for z/OS Client to talk to the Security Key Lifecycle Manager for z/OS Server. It acts as a Secure Sockets client.

Required
Yes.

TransportListener.ssl.keystore.password = password
Password to access TransportListener.ssl.keystore.name. The value for this property is obfuscated for additional security and the stanza name itself in the properties file is replaced with a new stanza that is named TransportListener.ssl.keystore.password.obfuscated.

Required
Optional.

TransportListener.ssl.keystore.type = jceks

Required
Optional.

Values

JCEKS | JCECCAKS |

JCERACFKS | JCECCARACFKS

TransportListener.ssl.port = *value*

Port the Security Key Lifecycle Manager for z/OS server listens on for requests from other Security Key Lifecycle Manager for z/OS Servers.

Required

Yes.

Values

Port number, 443 for example. The value must match the TransportListener.ssl.port property in the configuration properties file.

TransportListener.ssl.protocols = SSL_TLS

Security protocols

Required

Optional.

Values

SSL_TLS (default) | SSL | TLS

TransportListener.ssl.timeout = 10

Specifies how long socket waits on a read() before throwing a SocketTimeoutException.

Required

Optional.

Value Specified in minutes.

Default

10

TransportListener.ssl.truststore.name = *value*

The name of the database of public keys and signed certificates used to verify the identities of other clients and servers. If the TransportListener.ssl.clientauthentication property is **not** set to the default value of 0, then the Security Key Lifecycle Manager for z/OS Server, acting as the Secure Socket Server, must authenticate the client by using this file. This truststore is also used by the Security Key Lifecycle Manager for z/OS Client. It is used to talk to the Security Key Lifecycle Manager for z/OS Server and act as a Secure Sockets client.

Required

Yes.

TransportListener.ssl.truststore.type = jceks

Required

Optional.

Values

JCEKS | JCECCAKS | JCERACFKS | JCECCARACFKS

TransportListener.tcp.port = *value*

Port the Security Key Lifecycle Manager for z/OS server listens on for requests from tape drives. The default TCP port number is 3801.

Required
Yes.

Values
Port number, 10 for example.

TransportListener.tcp.timeout = *value*
Specifies how long a socket waits on a read() before throwing a SocketTimeoutException.

Required
Optional.

Values
Specified in minutes. 0 means no timeout.

Default
10

useSKIDefaultLabels = *value*
Specifies whether the Security Key Lifecycle Manager for z/OS creates Externally Encrypted Data Keys (EEDKs) for encrypted TS1120, TS1130,TS1140, or DS8000. It specifies if those EEDKS are created using the X509 Subject Key Identifier (SKI) hash instead of the label when a default alias, either from the device table or configuration file, is used. When a value of **true** is specified, the EEDKs are created using the SKI hash on the certificate. If this property is set to a value of **false** (or any other value, or left unspecified), the default label of a certificate is used.

Note: This property has no function for LTO tapes.

Required
Optional.

Values
true | false

Default
false

zOSCompatibility = *value*
The zOSCompatibility flag is used to identify the crypto capabilities of the z/OS system being used. This flag is typically used when hardware cryptography is being used on z/OS, ICSF. At one point, ICSF did not support the AES algorithm that Security Key Lifecycle Manager for z/OS uses and this flag was a work around for that issue. However, ICSF does support AES now, so this flag does not need to be used anymore.

Note: If you need to have the zOSCompatibility flag turned on one system, make sure that you have it turned on all systems that are serving keys to the same devices.

Required
Optional.

Note: This property does not need to be used.

Values

true The encrypted tape can be read by an instance of Security Key Lifecycle Manager for z/OS running on the z/OS Platform.

false Default. The encrypted tape can only be read by an instance of Security Key Lifecycle Manager for z/OS running on the z/OS platform if the Security Key Lifecycle Manager for z/OS is running with Java 5.0. Instances of Security Key Lifecycle Manager for z/OS cannot use the z/OS encrypted key support as provided by ICSF and zSeries hardware assisted cryptography.

Appendix C. Quick Start Guides

This topic provides instructions to guide you in getting started with setting up Security Key Lifecycle Manager for z/OS and different storage devices.

Quick Start Guide for TS1120, TS1130, and TS1140 Tape Drives

Get started with a basic configuration for TS1120, TS1130, and TS1140 Tape Drives.

The Security Key Lifecycle Manager for z/OS works with IBM encryption-enabled tape drives and system storage devices. The product helps in generating, protecting, storing, and maintaining encryption keys that are used to encrypt information being written to and decrypt information being read from devices.

This document shows how quickly you can install Security Key Lifecycle Manager for z/OS with TS1120, TS1130, and TS1140 tape drives and how easy it can be to set up and deploy. Because the JCEKS keystore type is the easiest and most transportable of the keystores supported, the steps below use this keystore type. If you want more information about a particular step or other supported keystore types, see the "Which Keystore is Right for You" on page 36.

Using Security Key Lifecycle Manager for z/OS with TS1120, TS1130, and TS1140 Tape Drives

Get started with using Security Key Lifecycle Manager for z/OS with TS1120, TS1130, and TS1140 Tape Drives.

Before you begin

Ensure you have the following:

- Java 5.0 SR5 or 6.0 GA and above. Java is available at http://www-03.ibm.com/systems/z/os/zos/tools/java/
- The unrestricted policy files for Java. The files are available at https://www14.software.ibm.com/webapp/iwm/web/preLogin.do?source=jcesdk.

Procedure

1. Install the Security Key Lifecycle Manager for z/OS as instructed in the Program Directory document. See, Program Directory for IBM Security Key Lifecycle Manager for z/OS. The following files are installed:
   - `/usr/lpp/ISKLM/samples/ISKLMConfig.properties.zos` - sample configuration file.
   - `hlq.SCKLSAMP(ISKLM)` - sample file that can be used to configure Security Key Lifecycle Manager for z/OS to work with JZOS.
   - `hlq.SCKLSAMP(ISKLMENV)` - sample file that can be used to configure Security Key Lifecycle Manager for z/OS to work with JZOS.
   - `hlq.SCKLSAMP(CKLJSMF)` - sample JCL to extract SMF records to an XML file.
   - `CKLENV (ISKLMENV)` - sample shell script file that includes configuration variables to run Security Key Lifecycle Manager for z/OS with Java 5.0.
   - `CLIPROC (ISKLM)` - sample file to run Security Key Lifecycle Manager for z/OS under JZOS.

- CKLJSMF (ISKLMSMF) - sample JCL to unload SMF audit records.

2. Create a directory /u/isklmsrv.
3. Change directory to /u/isklmsrv.
4. Copy the sample file /usr/lpp/ISKLM/samples/ISKLMConfig.properties.zos to /u/isklmsrv.
5. Ensure that your path is set to Java.
6. Create a JCEKS Keystore.

   Security Key Lifecycle Manager for z/OS needs a keystore with a certificate and private key. This certificate will be used to secure communications among the Security Key Lifecycle Manager for z/OS Servers as well as protecting the data key on the tape. This keytool command creates a new JCEKS keystore called ISKLMKeys.jck and populates it with a certificate and private key with the alias of isklmcert. This certificate will be valid for one year.

   a. Use the command:

      ```
      keytool -keystore ISKLMKeys.jck -storetype jceks -genkey -alias
      isklmcert -keyAlg RSA -keysize 2048 –validity 365 -storepass
      "somesecretphrase"
      ```

      When you issue this command, it prompts you for information it uses to create a distinguished name to put in the certificate. The prompts, with sample responses, look similar to these:

      ```
      What is your first and last name? [Unknown]: isklmcert
      What is the name of your organizational unit? [Unknown]: ISKLM
      What is the name of your organization? [Unknown]: IBM
      What is the name of your City or Locality? [Unknown]: Austin
      What is the name of your State or Province? [Unknown]: TX
      What is the two-letter country code for this unit? [Unknown]: US
      Is CN=isklmcert, OU=ISKLM, O=IBM, L=Austin, ST=TX, C=US correct?(type "yes" or "no"):
      ```

   b. Type yes and press Enter.

7. Modify the Security Key Lifecycle Manager for z/OS Server Configuration Properties File.

   Security Key Lifecycle Manager for z/OS requires two properties to be set based on the alias used in the previous step.

   a. Edit the ISKLMConfig.properties.zos file and add the following entries:

      ```
      drive.default.alias1 = isklmcert

      drive.default.alias2 = isklmcert

      drive.acceptUnknownDrives=true
      ```

   b. Save and close the ISKLMConfig.properties.zos file.

8. Add /usr/lpp/ISKLM/IBMSKLM.jar to your classpath.
9. Start theSecurity Key Lifecycle Manager for z/OS server using this command: `java com.ibm.ltklm.ISKLMServer /u/isklmsrv/ISKLMConfig.properties.zos`

Quick Start Guide for LTO Ultrium 4 and LTO Ultrium 5

Get started with a basic configuration for LTO Ultrium 4 and LTO Ultrium 5.

The Security Key Lifecycle Manager for z/OS works with IBM encryption-enabled tape drives and system storage devices. The product helps in generating, protecting, storing, and maintaining encryption keys that are used to encrypt information being written to and decrypt information being read from devices.

This document shows how quickly you can install Security Key Lifecycle Manager for z/OS with LTO Ultrium 4 and LTO Ultrium 5 and how easy it can be to set up and deploy. Because the JCEKS keystore type is the easiest and most transportable

of the keystores supported, the steps below use this keystore type. If you want more information about a particular step or other supported keystore types, see the "Which Keystore is Right for You" on page 36.

Using Security Key Lifecycle Manager for z/OS with LTO Ultrium 4 and LTO 5

The following instructions describe how to get started with using Security Key Lifecycle Manager for z/OS with LTO Ultrium 4 and LTO Ultrium 5.

Before you begin

Ensure you have the following:

- Java 5.0 SR5 or 6.0 GA and above. Java is available at http://www-03.ibm.com/systems/z/os/zos/tools/java/
- The unrestricted policy files for Java. The files are available at https://www14.software.ibm.com/webapp/iwm/web/preLogin.do?source=jcesdk.

Procedure

1. Install the Security Key Lifecycle Manager for z/OS as instructed in the Program Directory document. See, Program Directory for IBM Security Key Lifecycle Manager for z/OS. The following files will be installed:
   - `/usr/lpp/ISKLM/samples/ISKLMConfig.properties.zos` - sample configuration file.
   - `hlq.SCKLSAMP(ISKLM)` - sample file that can be used to configure Security Key Lifecycle Manager for z/OS to work with JZOS.
   - `hlq.SCKLSAMP(ISKLMENV)` - sample file that can be used to configure Security Key Lifecycle Manager for z/OS to work with JZOS.
   - `hlq.SCKLSAMP(CKLJSMF)` - sample JCL to extract SMF records to an XML file.
   - `CKLENV (ISKLMENV)` - sample shell script file that includes configuration variables to run Security Key Lifecycle Manager for z/OS with Java 5.0.
   - `CLIPROC (ISKLM)` - sample file to run Security Key Lifecycle Manager for z/OS under JZOS.
   - `CKLJSMF (ISKLMSMF)` - sample JCL to unload SMF audit records.
2. Create a directory `/u/isklmsrv`.
3. Change directories to `/u/isklmsrv`.
4. Copy sample file `/usr/lpp/ISKLM/samples/ISKLMConfig.properties.zos` to `/u/isklmsrv`.
5. Ensure that your path is set to Java.
6. Create a JCEKS Keystore.

   Security Key Lifecycle Manager for z/OS needs a keystore with a certificate and private key. This certificate will be used to secure communications among the Security Key Lifecycle Manager for z/OS Servers as well as protecting the data key on the tape. This keytool command creates a new JCEKS keystore called ISKLMKeys.jck and populates it with a certificate and private key with the alias of isklmcert. This certificate will be valid for one year.

   a. Use the command:

      ```
      keytool -keystore ISKLMKeys.jck -storetype jceks -genkey -alias
      isklmcert -keyAlg RSA -keysize 2048 –validity 365 -storepass
      "somesecretphrase"
      ```

When you issue this command, it prompts you for information it uses to create a distinguished name to put in the certificate. The prompts, with sample responses, look similar to these:

```
What is your first and last name? [Unknown]: isklmcert
What is the name of your organizational unit? [Unknown]: ISKLM
What is the name of your organization? [Unknown]: IBM
What is the name of your City or Locality? [Unknown]: Austin
What is the name of your State or Province? [Unknown]: TX
What is the two-letter country code for this unit? [Unknown]: US
Is CN=isklmcert, OU=ISKLM, O=IBM, L=Austin, ST=TX, C=US correct?(type "yes" or "no"):
```

b. Type `yes` and press Enter.

7. Generate encryption keys.

   For LTO encryption, Security Key Lifecycle Manager for z/OS needs a number of symmetric keys to be pre-generated and stored in a keystore. This keytool command generates 32 256-bit AES keys and stores them in the keystore created in the previous step.

   a. Run this command from the Security Key Lifecycle Manager for z/OS directory to have the keystore file created in that directory.

      ```
      keytool –keystore ISKLMKeys.jck –storetype jceks –genseckey –keyAlg
      aes –keysize 256 –aliasrange key00-1f
      ```

      The resulting keys will have the names key00000000000000000 through key0000000000000001f.

      This command prompts you for a keystore password to access the keystore. Enter the desired password and press Enter. However, when prompted for a key password, just press Enter. Do not type in a new or different password. This will cause the key password to be the same as the keystore password. Please note the keystore password entered here as it will be needed later when starting the Security Key Lifecycle Manager for z/OS.

8. Modify the Security Key Lifecycle Manager for z/OS Server Configuration Properties File.

   a. Edit the ISKLMConfig.properties.zos file and add the following entries:

      ```
      symmetricKeySet=key00-1f
      ```

      ```
      drive.acceptUnknownDrives=true
      ```

   b. Save and close the ISKLMConfig.properties.zos file.

9. Add `/usr/lpp/ISKLM/IBMSKLM.jar` to your classpath.

10. Start theSecurity Key Lifecycle Manager for z/OS server using this command: `java com.ibm.ltklm.ISKLMServer /u/isklmsrv/ISKLMConfig.properties.zos`

Quick Start Guide for DS8000

Get started with a basic configuration for DS8000.

The Security Key Lifecycle Manager for z/OS works with IBM encryption-enabled tape drives and system storage devices. The product helps in generating, protecting, storing, and maintaining encryption keys that are used to encrypt information being written to and decrypt information being read from devices.

This document shows how quickly you can install Security Key Lifecycle Manager for z/OS with DS8000 and how easy it can be to set up and deploy. Because the JCEKS keystore type is the easiest and most transportable of the keystores supported, the steps below use this keystore type. If you want more information about a particular step or other supported keystore types, see the “Which Keystore is Right for You” on page 36.

Using Security Key Lifecycle Manager for z/OS with DS8000

The following instructions describe how to get started with using Security Key Lifecycle Manager for z/OS with DS8000.

Before you begin

Ensure you have the following:

- Java 5.0 SR5 or 6.0 GA and above. Java is available at http://www-03.ibm.com/systems/z/os/zos/tools/java/
- The unrestricted policy files for Java. The files are available at https://www14.software.ibm.com/webapp/iwm/web/preLogin.do?source=jcesdk.

Procedure

1. Install the Security Key Lifecycle Manager for z/OS as instructed in the Program Directory document. See, Program Directory for IBM Security Key Lifecycle Manager for z/OS. The following files will be installed:
   - `/usr/lpp/ISKLM/samples/ISKLMConfig.properties.zos` - sample configuration file.
   - `hlq.SCKLSAMP(ISKLM)` - sample file that can be used to configure Security Key Lifecycle Manager for z/OS to work with JZOS.
   - `hlq.SCKLSAMP(ISKLMENV)` - sample file that can be used to configure Security Key Lifecycle Manager for z/OS to work with JZOS.
   - `hlq.SCKLSAMP(CKLJSMF)` - sample JCL to extract SMF records to an XML file.
   - `CKLENV (ISKLMENV)` - sample shell script file that includes configuration variables to run Security Key Lifecycle Manager for z/OS with Java 5.0.
   - `CLIPROC (ISKLM)` - sample file to run Security Key Lifecycle Manager for z/OS under JZOS.
   - `CKLJSMF (ISKLMSMF)` - sample JCL to unload SMF audit records.
2. Create a directory `/u/isklmsrv`.
3. Change directories to `/u/isklmsrv`.
4. Copy sample file `/usr/lpp/ISKLM/samples/ISKLMConfig.properties.zos` to `/u/isklmsrv`.
5. Ensure that your path is set to Java.
6. Create a JCEKS Keystore.

   Security Key Lifecycle Manager for z/OS needs a keystore with a certificate and private key. This certificate will be used to secure communications among the Security Key Lifecycle Manager for z/OS Servers as well as protecting the data key on the tape. This keytool command creates a new JCEKS keystore called ISKLMKeys.jck and populates it with a certificate and private key with the alias of isklmcert. This certificate will be valid for one year.

   a. Use the command:

      ```
      keytool -keystore ISKLMKeys.jck -storetype jceks -genkey -alias
      isklmcert -keyAlg RSA -keysize 2048 –validity 365 -storepass
      "somesecretphrase"
      ```

      When you issue this command, it prompts you for information it uses to create a distinguished name to put in the certificate. The prompts, with sample responses, look similar to these:

      ```
      What is your first and last name? [Unknown]: isklmcert
      What is the name of your organizational unit? [Unknown]: ISKLM
      What is the name of your organization? [Unknown]: IBM
      What is the name of your City or Locality? [Unknown]: Austin
      ```

```
What is the name of your State or Province? [Unknown]: TX
What is the two-letter country code for this unit? [Unknown]: US
Is CN=isklmcert, OU=ISKLM, O=IBM, L=Austin, ST=TX, C=US correct?(type "yes" or "no"):
```

b. Type yes and press Enter.

7. Ensure the configuration file has the entry `ds8k.acceptUnknownDrives=true`. See, "Creating Security Key Lifecycle Manager for z/OS configuration file" on page 86.
8. Add `/usr/lpp/ISKLM/IBMSKLM.jar` to your classpath.
9. Start theSecurity Key Lifecycle Manager for z/OS server using this command: `java com.ibm.ltklm.ISKLMServer /u/isklmsrv/ISKLMConfig.properties.zos`

Appendix D. Frequently Asked Questions

Can some combination of application-based key management and system- or library-managed encryption be used?

No. Currently, Tivoli Storage Manager is the only application that provides application-managed tape encryption for TS1120, TS1130, and TS1140 tape drives. When application-managed encryption is used, the encryption is transparent at the system and library layers. Likewise, when system- or library-managed encryption is used, the process is transparent at the other layers. Each method of encryption management is exclusive of the others. For system-managed encryption and library-managed encryption, the applications need not be changed in any way.

Must the Security Key Lifecycle Manager for z/OS be installed and running on every system that might generate a request to encrypt or decrypt a tape?

For library- or system-managed encryption, the system where the tape drive write request originates need not be where the Security Key Lifecycle Manager for z/OS is running. An instance of Security Key Lifecycle Manager for z/OS does not have to be running on every system from which an encrypting tape drive is accessed.

Since RACF keyrings do not have a password, what should the following statements have coded for a password value?

Either omit these three statements from the configuration properties file or code them as follows:

config.keystore.password = password
TransportListener.ssl.keystore.password = password
TransportListener.ssl.truststore.password = password

Where password is, literally, password.

How do I know what keystore to choose?

The keystore choice is dependent upon the system on which the Security Key Lifecycle Manager for z/OS is running. Different platforms have different, platform-specific features (often tied to the hardware crypto support on those platforms). These differences manifest themselves in the keystore choice. When the platform where the Security Key Lifecycle Manager for z/OS runs is chosen with multiple Security Key Lifecycle Manager for z/OS running, then the keystore type must be chosen.

If I include the "drive.acceptUnknownDrives = True" parameter, should I still include the "config.drivetable.file.url = FILE:/filename" parameter in the configuration file?

config.drivetable.file.url must always be specified. It is where the drive information is placed. If you set `drive.acceptUnknownDrives = True`, specify the `drive.default.alias1` and `drive.default.alias2` variables to the correct certificate alias/key label.

Is FILE:/filename the correct syntax for the config.drivetable.file.url property? FILE:///filename appears in the sample file, and FILE:../ in the description.

The examples are correct. This syntax is a URL specification and is not what people normally expect for a directory structure specification.

Does the Security Key Lifecycle Manager for z/OS perform any Certificate Revocation List (CRL) checking?

No, the Security Key Lifecycle Manager for z/OS does not perform any CRL checking

What happens when the certificate being used to encrypt the tapes expires? Will the Security Key Lifecycle Manager for z/OS read previously encrypted tapes?

It does not matter toSecurity Key Lifecycle Manager for z/OS if the certificate has expired. It continues to honor these certificates and read previously encrypted tapes. However the expired certificate must remain in the keystore in order for previously encrypted tapes to be read or appended.

Will the Security Key Lifecycle Manager for z/OS require certificate renaming on renewal?

The Security Key Lifecycle Manager for z/OS is configured by default to honor new key requests with expired certificates. When the Security Key Lifecycle Manager for z/OS is configured this way certificate renewal is not required. If this function is disabled and this private key/certificate pair must still be used for new key requests, then the user must renew the certificate. The certificate alone (validity dates) would be renewed but not the associated keys.

Will later versions of Security Key Lifecycle Manager for z/OS still read the encrypted tapes created with earlier versions of the software?

Yes. The Security Key Lifecycle Manager for z/OS reads certificates regardless of release.

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:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
1623-14, Shimotsuruma, Yamato-shi
Kanagawa 242-8502 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 might 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
2Z4A/101
11400 Burnet Road
Austin, TX 78758 U.S.A.

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

The licensed program described in this document 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.

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

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.

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

All IBM prices shown are IBM's suggested retail prices, are current and are subject to change without notice. Dealer prices may vary.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

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.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. _enter the year or years_. All rights reserved.

If you are viewing this information in softcopy form, the photographs and color illustrations might not be displayed.

Trademarks

IBM, the IBM logo, and ibm.com® are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at http://www.ibm.com/legal/copytrade.shtml

Adobe, Acrobat, PostScript and all Adobe-based trademarks are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, other countries, or both.

IT Infrastructure Library is a registered trademark of the Central Computer and Telecommunications Agency which is now part of the Office of Government Commerce.

Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

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

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

ITIL is a registered trademark, and a registered community trademark of the Office of Government Commerce, and is registered in the U.S. Patent and Trademark Office.

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

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

Cell Broadband Engine is a trademark of Sony Computer Entertainment, Inc. in the United States, other countries, or both and is used under license therefrom.

Linear Tape-Open, LTO, the LTO Logo, Ultrium, and the Ultrium logo are trademarks of HP, IBM Corp. and Quantum in the U.S. and other countries.

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

Glossary

This glossary includes terms and definitions related to Security Key Lifecycle Manager for z/OS.

This glossary defines the special terms, abbreviations, and acronyms used in this publication and other related publications.

AES Advanced Encryption Standard. A block cipher adopted as an encryption standard by the US government.

alias See key label.

certificate
A digital document that binds a public key to the identity of the certificate owner, thereby enabling the certificate owner to be authenticated.

certificate label
See key label.

certificate store
See keystore.

DK Data Key. An alphanumeric string used to encrypt data.

EEDK Externally Encrypted Data Key. A Data Key that has been encrypted (wrapped) by a Key Encryption Key prior to being stored in the data cartridge. See KEK.

EEFMT2
Enterprise Encryption Format. AES 256-bit encrypted data written recorded at the performance and capacity format used by the native 3592 Model E05.

encryption
The conversion of data into a cipher. A key is required to encrypt and decrypt the data. Encryption provides protection from persons or software that attempt to access the data without the key.

KEK Key Encrypting Key. An alphanumeric, asymmetric key used to encrypt the Data Key. See EEDK.

key label
A unique identifier used to match the EEDK with the private key (KEK) required to unwrap the protected symmetric data key. Also called alias or certificate label depending on which keystore is used.

key ring
See keystore.

keystore
A database of private keys and their associated X.509 digital certificate chains used to authenticate the corresponding public keys. Also called certificate store or key ring in some environments.

PKDS Public Key Data Set. Also PKA cryptographic Key Data Set.

private key
One key in an asymmetric key pair, typically used for decryption. The Security Key Lifecycle Manager for z/OS uses private keys to unwrap protected AES data keys prior to decryption.

public key
One key in an asymmetric key pair, typically used for encryption. The Security Key Lifecycle Manager for z/OS uses public keys to wrap (protect) AES data keys prior to storing them on the tape cartridge.

rekey The process of changing the asymmetric Key Encrypting Key (KEK) that protects the Data Key (DK) stored on an already encrypted tape, thereby allowing different entities access to the data.

RSA Rivest-Shamir-Adleman algorithm. A system for asymmetric, public-key cryptography used for encryption and authentication. It was invented in 1977 by Ron Rivest, Adi Shamir, and Leonard Adleman. The security of the system depends on the difficulty of factoring the product of two large prime numbers.

Index

L

M

O

P

Q

R

S

T

U

X

Printed in USA

SC14-7628-00