Axel Buecker
William C. Johnston

IBM Security Key Lifecycle Manager for z/OS: Deployment and Migration Considerations

Overview

This IBM® Redpaper™ publication discusses IBM Security Key Lifecycle Manager (ISKLM) for IBM z/OS® V1.1 and includes topics that discuss encryption capabilities, installation considerations, keystores, auditing, troubleshooting, and migration considerations. We also discuss common practices for key management and provide a sample REXX code procedure for exporting a data key.

This IBM Redpaper publication contains the following sections:

Device-based encryption overview

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

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

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

Security Key Lifecycle Manager for z/OS also supports the IBM DS8000® Storage Controller. This support requires the appropriate microcode bundle version on the DS8000 Storage Controller, Licensed Internal Code (LIC) level 64.2 or higher.

Encryption

Tape write requests flow from the requesting systems to the Automated Tape Libraries and tape drives. The decision to encrypt is made either at the library or at the system. After the decision to encrypt is made, that indication is sent to the tape drive. The tape drive makes a request to the key manager for an encryption key across a secure network session. The data encryption key is delivered to the tape drive. The data encryption key is installed into the encryption engine on the tape drive. The I/O stream to the drive is then compressed, encrypted, and written to the tape.

Decryption

Tape read requests flow from the requesting systems to the tape drive. The decision to decrypt is made at the drive based on whether the cartridge is encrypted. The tape drive makes a request to the key manager for the decryption key across a secure network session. The key manager retrieves the key from the keystore and delivers it to the drive. The data decryption key is installed into the decryption engine. The data is then read from the drive, decrypted, decompressed, and sent up the channel path. After the cartridge is unmounted, the key is discarded by the drive.

LTO encryption

TS1040 is LTO-4 compliant and so uses pre-generated symmetric keys for data encryption. An administrator generates multiple Advanced Encryption Standard (AES) data keys and stores them directly in the keystore.

As scratch tapes are mounted, the key manager selects a key from the keystore for distribution to the device. The same key is used for all data on the cartridge. The label of this key is written to the media and also stored in cartridge memory. Subsequent read and write operations require a connection to a key manager where the specific key resides under the same label. Keys can reside in multiple key manager keystores.

3592 encryption

When contacted to deliver an encryption key for 3592 devices, the key manager creates a unique 256-bit AES symmetric key using a random number generator. This key is sent to the device. The symmetric data key is encrypted with a public keypair, which is stored in the key manager. This encrypted data key is written to the media and also stored in cartridge memory. For 3592 encryption, individual datakeys are not stored by the key manager in its keystore; only the keypairs are stored. Subsequent read and write operations require a connection to a key manager where the specific keypair resides. The TS1130 sends the encrypted data key to the key manager, which decrypts and returns the data key back to the device over a secure session. Keypairs can reside in multiple key manager keystores.

IBM Security Key Lifecycle Manager overview

IBM Security Key Lifecycle Manager for z/OS V1.1 addresses the dynamics that make key lifecycle management a vital initiative:

- Supports storage management solutions: Encryption is a critical capability for storage management, and effective key lifecycle management must be a critical control point in a computing infrastructure.
- Helps reduce the cost of data loss: The costs that are associated with the loss of data can be significant. Clearly, investment now to develop a data security strategy can help avoid the much higher costs that are associated with cleaning up after a breach. To get ahead of this data breach risk, encryption and key lifecycle management need to be high on clients' priority lists.
- Includes Security Key Lifecycle Manager for z/OS V1.1 enhancements:
  - Removes the dependency on the system service runtime environment (SSRE) and IBM DB2®, making the migration from the IBM Encryption Key Manager (EKM) (key server for the IBM System Storage® TS1120, TS1130, and IBM LTO Ultrium 4 tape drives) and the installation of Security Key Lifecycle Manager for z/OS simpler.
  - Serves keys for data encryption with IBM System Storage TS1120, TS1130, TS1140, DS8000 Turbo drive, IBM LTO Ultrium 4, and IBM LTO Ultrium 5 tape drives. It provides the administration and maintenance of the keys and allows for automatically adding drives to the device table.
  - Supports System Management Facility (SMF) for audit records.

Installation considerations on z/OS

When installing IBM Security Key Lifecycle Manager on z/OS, it is important to prepare Java for IBM Security Key Lifecycle Manager on z/OS. It is equally important to be consistent in the configuration of the IBM Security Key Lifecycle Manager for z/OS environment. Various settings are made when implanting hardware or non-hardware-based keystores.

File placement

Several files must be moved for IBM Security Key Lifecycle Manager to operate correctly on z/OS. The Java Virtual Machine Load Module, JVMLDMxx, is copied to the z/OS link list concatenation, security files are placed in the proper Java directory, and the IBM Security Key Lifecycle Manager for z/OS jar must be accessible to the Java virtual machine (JVM).

The Java Virtual Machine Load Module

Copy the Java Virtual Machine Load Module JVMLDMxx into the linklist concatenation:

```
cp -X ./mvstools/JVMLDM60 "//'SYS1.SIEALNKE(JVMLDM60)'"
```

Java Security Policy Files

Copy unrestricted policy files into Java's security directory:

```
cp $JAVA_HOME/demo/jce/policy-files/unrestricted/*    $JAVA_HOME/lib/security
```

IBMSKLM for z/OS jar

Copy the `IBMSKLM` jar file into the Java library extensions:

```
cp /usr/lpp/ISKLM/IBMSKLM.jar $JAVA_HOME/lib/ext
```

Configuration consistency

It is important to be consistent in the configuration of the IBM Security Key Lifecycle Manager for z/OS environment. Various settings are made when implanting hardware or non-hardware-based keystores. The settings across the JzOS environment file, the IBM Security Key Lifecycle Manager configuration file, and the java.security API providers list must match.

Software-based JCERACFKS

Software-based JCERACFKS keystores need a non-hardware cryptographic services provider. The provider is listed in the JzOS environment file whose location is indicated by the started task procedure STDENV DD card:

```
//STDENV DD DSN=USER.PARMLIB(CKLENV),DISP=SHR

# for JCERACFKS, following IJO definition is required:

IJO="-Djava.protocol.handler.pkgs=com.ibm.crypto.provider"
```

The Java cryptographic services providers that are listed in `$JAVA_HOME/security/java.security` also need to include the IBMJCE provider:

```
security.provider.2=com.ibm.crypto.provider.IBMJCE
```

The keystores defined in the configuration file and indicated by the started task procedure STDENV DD card must all be of the same type. The following line points to the IBM Security Key Lifecycle Manager configuration properties file:

```
export ISKLMARGS="/ftssuser/SKLMSRV/config/ISKLMConfig.properties.zos"
```

Example 1 shows the properties within the configuration file that denote the type of keystores in use.

Example 1 Keystore properties

```
config.keystore.type = JCERACFKS
Admin.ssl.keystore.type = JCERACFKS
Admin.ssl.truststore.type = JCERACFKS
TransportListener.ssl.truststore.type = JCERACFKS
```

Additionally, you must configure the correct service provider, as shown in Example 2.

Example 2 Keystore providers

```
Admin.ssl.keystore.provider = IBMJCE
TransportListener.ssl.truststore.provider = IBMJCE
config.keystore.provider = IBMJCE
Admin.ssl.truststore.provider = IBMJCE
TransportListener.ssl.keystore.provider = IBMJCE
```

Hardware-based JCECCAKS and JCECCARACFKS

Hardware-based JCECCARACFKS keystores need a hardware cryptographic services provider listed in the JzOS environment file:

```
# for JCECCARACFKS, following IJO definition is required

IJO="-Djava.protocol.handler.pkgs=com.ibm.crypto.hdwrCCA.provider"
```

The Java cryptographic services providers that are listed in `$JAVA_HOME/security/java.security` also need to include the IBMJCECCA provider:

```
security.provider.2=com.ibm.crypto.hdwrCCA.provider.IBMJCECCA
```

The keystores defined in the configuration file and indicated by the started task procedure STDENV DD card must all be of the same type. The following line points to the IBM Security Key Lifecycle Manager configuration properties file:

```
export ISKLMARGS="/ftssuser/SKLMSRV/config/ISKLMConfig.properties.zos"
```

Example 3 shows the properties within the configuration file that denote the type of keystores in use.

Example 3 Keystore properties

```
config.keystore.type = JCECCARACFKS
Admin.ssl.keystore.type = JCECCARACFKS
Admin.ssl.truststore.type = JCECCARACFKS
TransportListener.ssl.truststore.type = JCECCARACFKS
```

Additionally, you must configure the correct service provider, as shown in Example 4.

Example 4 Keystore providers

```
Admin.ssl.keystore.provider = IBMJCECCA
TransportListener.ssl.truststore.provider = IBMJCECCA
config.keystore.provider = IBMJCECCA
Admin.ssl.truststore.provider = IBMJCECCA
TransportListener.ssl.keystore.provider = IBMJCECCA
```

Keystore options

Keys and certificates are accessed in Java through the Java Cryptographic Element (JCE). The keystore that is selected must support the types of keys that are required by the devices. The 3592 device-based encryption uses wrapper keys. The keystore holds the key wrapping key in the form of X.509v3 digital certificates. The LTO drives require an unwrapped key, so the keystore must hold the actual 256-bit AES key. Specific keystores work in conjunction with Integrated Cryptographic Services Facility (ICSF) to provide the protection of the mainframe cryptographic hardware.

Four types of keystores are available:

- JCEKS: For all types of keys. A password-protected UNIX-based file is the key repository.
- JCERACFKS: Can only hold certificates. Certificates and keys exist in a System Authorization Facility (SAF)-based keyring.

- JCECCAKS: For all types of keys. A UNIX-based file points to keys residing in ICSF's cryptographic key dataset (CKDS).
- JCECCARACFKS: Can only hold certificates. Certificates reside in a SAF-based keyring; keys reside in ICSF's public key dataset (PKDS).

Table 1 shows the various keystore types and their relationships to the key repository.

Table 1 Keystore types

Name	Recommended Key management tool	Key backing store type	Platforms	Hardware Crypto?
JCEKS	Keytool	UNIX file	All UNIX	No
JCERACFKS	RACF/ACF2/Top Secret	Proprietary database	z/OS	No
JCECCAKS	Hwkeytool with ICSF	UNIX file and ICSF	z/OS	Yes
JCECCARACFKS	RACF/ACF2/Top Secret with ICSF	Proprietary database and ICSF	z/OS	Yes

Table 2 pairs the Java Cryptographic Provider with the appropriate Java keystores.

Table 2 Java cryptographic providers

Provider	Valid keystores/truststores
IBMJCE	JCEKS JCERACFKS
IBMJCECCA	JCECCAKS JCECCARACFKS

Java file-based keystore: JCEKS

Use this keystore type if you only use Java software. For all operating systems and a 3592 tape drive, LTO tape drive, or DS8000 Turbo drive, ensure that the flat file JCEKS keystore resides in a restricted area of the file system on the IBM Tivoli® Key Lifecycle Manager system. Use a JCEKS keystore for all operating systems other than z/OS. You might also use this keystore type on a z/OS system if you want to use JCE software and a flat file to store keys.

Implementing JCEKS

Keytool is a key and certificate management utility. It enables users to administer their own public/private key pairs and associates. The keytool utility stores the keys and certificates in a keystore. The JCEKS deployment implements the keystore as a file. It protects private keys with a password.

Keytool is part of the Java deployment. It resides in the `/usr/lpp/java/J5.0/bin/keytool` bin directory.

The command to generate the AES keys is **–genseckey**. Example 5 on page 7 shows the complete usage for the command.

Example 5 -genseckey command

```
keytool -genseckey {-alias alias | -aliasrange aliasRange} {-keyalg keyalg}
{-keysize keysize} {-keypass keypass} {-storetype storetype} {-keystore keystore}
[-storepass storepass] [-providerClass provider_class_name] {-v} {-Jjavaoption}
```

The command to generate a Rivest-Shamir-Adleman algorithm (RSA) keypair using keytool is **–genkey**. Example 6 shows the complete usage for the command.

Example 6 -genkey command

```
keytool –genkey {-alias alias} {-keyalg keyalg} {-keysize keysize} {-sigalg
sigalg} [-dname dname] [-keypass keypass] {-validity valDays} {-storetype
storetype} {-keystore keystore} [-storepass storepass] [-provider
provider_class_name] {-v} {-Jjavaoption}
```

For more information, refer to the following website:

http://www.ibm.com/developerworks/java/jdk/security/50/secguides/keytoolDocs/KeyToolUserGuide-150.html

SAF-based keystore: JCERACFKS

JCERACFKS is a keystore that uses keys on a keyring in the SAF external security manager (ESM). Examples of an ESM are IBM RACF® from IBM, and ACF2 and Top Secret from Computer Associates.

Use this keystore type to store key material in your ESM keyring that does not use ICSF. JCERACFKS keystores are compatible with both IBMJCECCA and IBMJCE, that is, with both hardware and software providers.

If you use an ESM keyring for the master keystore, you will need to give the IBM Security Key Lifecycle Manager-started task ID user access to that keyring before you start the key manager.

The command to generate an RSA keypair using RACF is RACDCERT GENCERT.

LTO restrictions: A SAF-based keyring cannot be used to store symmetric keys for LTO tape drives.

Implementing JCERACFKS

Use the following procedure to allow IBM Security Key Lifecycle Manager to use a JCERACFKS keystore. Assume that the IBM Security Key Lifecycle Manager started task user ID is SKLMSRV and the case-sensitive keyring name is ISKLMRing:

1. Configure Java to use the correct service providers. This example assumes that Java 6.0 is installed at `/usr/lpp/java/J6.0`. Ensure that the IBMJCE Java Security Service Provider is listed in `/usr/lpp/java/J6.0/lib/security/java.security`. See Example 7.

   Example 7 Procedure to use a JCERACFKS keystore

   ```
   #.....................................................................
   ........................
   #This example assumes Java 6.0
   # List of providers and their preference orders (see above):
   ```

```
#.....................................................................
.......................
security.provider.1=com.ibm.jsse2.IBMJSSEProvider2
security.provider.2=com.ibm.crypto.provider.IBMJCE
security.provider.3=com.ibm.security.jgss.IBMJGSSProvider
security.provider.4=com.ibm.security.cert.IBMCertPath
security.provider.5=com.ibm.security.sasl.IBMSASL
```

2. Perform the SAF keyring operations for IBM Security Key Lifecycle Manager. These commands assume that RACF is the installed external security manager on z/OS.

   a. Add the keyring for encryption certificates using RACDCERT:

   ```
   RACDCERT ID(SKLMSRV) ADDRING(ISKLMRing)
   ```

   b. Add the keyring for Secure Sockets Layer (SSL) communications using RACDCERT:

   ```
   RACDCERT ID(SKLMSRV) ADDRING(ISKLMSSL)
   ```

   c. Add an x.509v3 digital certificate to the RACF keyring:

   ```
   RACDCERT GENCERT
   ID(SKLMSRV)
   SUBJECTSDN(CN('Key Encrypting Key for device based encryption'))
   SIZE(4096)
   NOTBEFORE( DATE( date cert_becomes_valid ))
   NOTAFTER( DATE( date_cert_expires ))
   WITHLABEL('Use.An.Agreed.Upon.Naming.Convention')
   ```

   d. Add an x.509v3 digital certificate to the RACF keyring:

   ```
   RACDCERT GENCERT
   CERTAUTH
   SUBJECTSDN(CN('ISKLM CA'))
   SIZE(4096)
   NOTBEFORE( DATE( date cert_becomes_valid ))
   NOTAFTER( DATE( date_cert_expires ))
   WITHLABEL('ISKLM.CA')
   ```

   e. Connect the certificates to the keyring:

   i.
   ```
   RACDCERT ID(SKLMSRV)
   CONNECT(ID(SKLMSRV)
   LABEL('Use.An.Agreed.Upon.Naming.Convention')
   RING(ISKLMRing)
   USAGE(PERSONAL))
   ```

   ii. It is also necessary to connect a CERTAUTH certificate to the keyring:

   ```
   RACDCERT ID(SKLMSRV)
   CONNECT(ID(SKLMSRV)
   LABEL('ISKLM.CA')
   RING(ISKLMRing)
   USAGE(CERTAUTH))
   ```

   iii. Connect a CERTAUTH certificate to the SSL keyring:

   ```
   RACDCERT ID(SKLMSRV)
   CONNECT(ID(SKLMSRV)
   ```

```
LABEL('ISKLM.CA')
RING(ISKLMSSL)
USAGE(CERTAUTH))
```

3. Configure IBM Security Key Lifecycle Manager to use a SAF keyring. You must consider several keyring-related configuration options:
   a. `Admin.ssl.keystore.name`:
      - This database contains the key pairs and certificates that are used for Secure Sockets 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 Layer client presents to the Secure Sockets Layer server comes from this keystore.
      - This keystore is optional but advised.
      - An example is `Admin.ssl.keystore.name = safkeyring://SKLMSRV/ISKLMSSL`.
   b. `Admin.ssl.truststore.name`:
      - This database file is used to check the trust of the Secure Sockets Layer server certificate that the server presents to the Secure Sockets Layer client.
      - This keystore is optional but advised.
      - An example is `Admin.ssl.truststore.name = safkeyring://SKLMSRV/ISKLMSSL`.
   c. `config.keystore.file`:
      - This file specifies the keystore to be used to store device-based encryption keys and certificates.
      - This keystore is required.
      - An example is `config.keystore.file = safkeyring://SKLMSRV/ISKLMRing`.
   d. `TransportListener.ssl.keystore.name`:
      - This database is used by the Security Key Lifecycle Manager for z/OS server to hold the certificate and private keys for the Secure Sockets Layer server. This certificate is given to the Secure Sockets Layer 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 Layer client.
      - This keystore is required.
      - An example is `TransportListener.ssl.keystore.name = afkeyring://SKLMSRV/ISKLMSSL`.
   e. `TransportListener.ssl.truststore.name`:
      - This database contains the public keys and signed certificates that are 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, the Security Key Lifecycle Manager for z/OS server, acting as the Secure Sockets Layer 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 Layer client.
      - This keystore is required.
      - An example is `TransportListener.ssl.truststore.name = afkeyring://SKLMSRV/ISKLMSSL`.

4. The IBM Security Key Lifecycle Manager Started Task user ID must have the proper authority to key management activities:
   a. Read the keyring:

      i. RDEFINE FACILITY IRR.DIGTCERT.LIST UACC(NONE)

      ii.PERMIT IRR.DIGTCERT.LIST CL(FACILITY) ID(SKLMSRV) ACCESS(READ)

   b. Read the keys on the keyring:

      i. RDEFINE FACILITY IRR.DIGTCERT.LISTRING UACC(NONE)

      ii.PERMIT IRR.DIGTCERT.LISTRING CL(FACILITY) ID(SKLMSRV) ACCESS(READ)

5. The Security Administrator must have the proper authority to key management activities:
   a. Read and write to the IBM Security Key Lifecycle Manager keyrings:

      i. RDEFINE RDATALIB SKLMSRV.ISKLMRing.UPD UACC(NONE)

      ii.DEFINE RDATALIB SKLMSRV.ISKLMSSL.UPD UACC(NONE)

      iii.PERMIT SKLMSRV.ISKLMRing.UPD ID(<Security_Admin user ID>) CL(RDATALIB) ACCESS(READ)

      iv.PERMIT SKLMSRV.ISKLMSSL.UPD ID(<Security_Admin user ID>) CL(RDATALIB) ACCESS(READ)

ICSF: Cryptographic hardware keystores

Two types of keystores are available to take advantage of cryptographic hardware: JCECCAKS, which uses a UNIX-based file to point to the ICSF CKDS and PKDS, and JCERACFKS, which holds certificates on a keyring with pointers to keys in ICSF PKDS.

ICSF-only keystore: JCECCAKS

JCECCAKS uses the Java tool hwkeytool to manage key material. JCECCAKS manages symmetric and asymmetric keys residing in ICSF. The keystore file in z/OS UNIX System Services contains the key labels that are used by ICSF to access protected keys (Figure 1 on page 10).

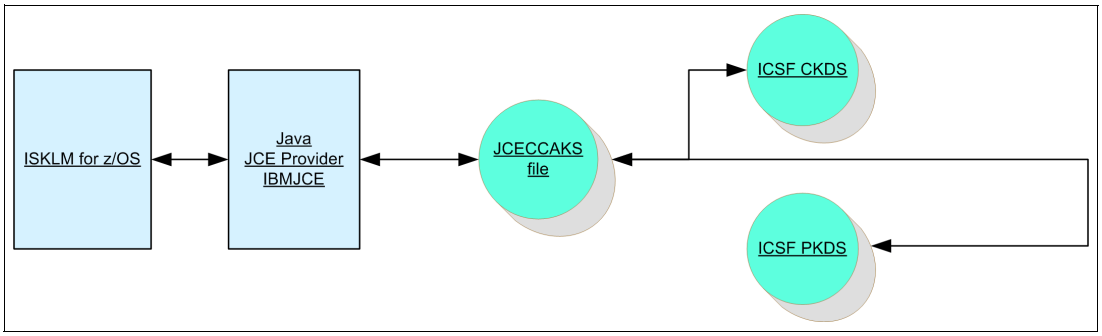

Figure 1 ICSF-only keystore

Implementing JCECCAKS

The Java tool **hwkeytool** is part of the Java deployment. It resides in the bin directory:

```
/usr/lpp/java/J5.0/bin/hwkeytool
```

The command to generate AES keys is **–genseckey**. Example 8 shows the complete usage for the command.

Example 8 -genseckey command

```
hwkeytool -genseckey {-alias alias | -aliasrange aliasRange} {-keyalg keyalg}
{-keysize keysize} {-keypass keypass} {-storetype storetype} {-keystore keystore}
[-storepass storepass] [-providerClass provider_class_name] {-v} {-Jjavaoption}
```

The command to generate an RSA keypair using keytool is **–genkey**. Example 9 shows the complete usage for the command.

Example 9 -genkey command

```
hwkeytool -genkey {-alias alias} {-keyalg keyalg} {-keysize keysize} {-sigalg
sigalg} [-dname dname] [-keypass keypass] {-validity valDays} {-storetype
storetype} {-keystore keystore} [-storepass storepass] {-keylabel keylabel}
{-hardwaretype hardwaretype} {-hardwareusage hardwareusage} [-provider
provider_class_name] {-v} {-Jjavaoption}
```

For more information, go to the following website:

ftp://ftp.software.ibm.com/s390/java/jce4758/hwkeytool.html

RACF and ICSF keystore: JCECCARACFKS

JCECCARACFKS is a SAF keyring/ICSF-based keystore that is supported on the z/OS operating system only. This keystore uses certificates that are generated in a RACF or SAF equivalent where the key material is stored in ICSF. The JCECCARACFKS keystore makes use of all the security advantages of both RACF/SAF and ICSF/CryptoExpress ( ).

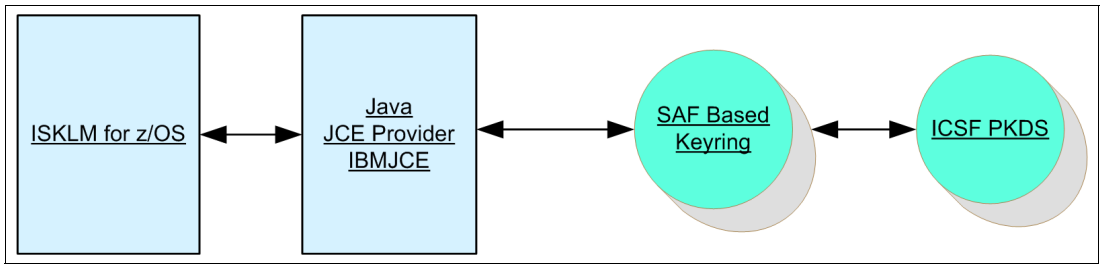

Figure 2 RACF and ICSF keystore

LTO restrictions: You cannot use SAF-based keystores to manage symmetric keys for LTO devices. No connection exists to the ICSF CKDS through JCECCARACFKS.

You must set the hardware JCE provider in the Java security properties file. Use this keystore type to store key material in your RACF keyring that uses ICSF for a z/OS operating system with a 3592 tape drive or DS8000 Turbo drive.

If you use a RACF keyring for the master keystore, you will need to give the IBM Security Key Lifecycle Manager started task ID user access to that RACF keyring before you select and configure the RACF keyring using a JCECCARACFKS.

Implementing JCECCARACFKS

Use the following procedure to allow IBM Security Key Lifecycle Manager to use a JCECCARACFKS keystore. Assume that the IBM Security Key Lifecycle Manager STC user ID is SKLMSRV and the case-sensitive keyring name is ISKLMRing.

1. Configure Java to use the correct service providers. This example assumes that Java 6.0 is installed at `/usr/lpp/java/J6.0`. Ensure that the IBMJCECCA Java Security Service Provider is listed in `/usr/lpp/java/J6.0/lib/security/java.security`. See Example 10.

Example 10 Procedure to use a JCECCARACFKS keystore

```
#.....................................................................
.......................
#This example assumes Java 6.0
# 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
```

2. Perform the SAF keyring operations for IBM Security Key Lifecycle Manager. These commands assume that RACF is the installed external security manager on z/OS.
   a. Add the keyring for encryption certificates using RACDCERT:

      ```
      RACDCERT ID(SKLMSRV) ADDRING(ISKLMRing)
      ```

   b. Add the keyring for SSL communications using RACDCERT:

      ```
      RACDCERT ID(SKLMSRV) ADDRING(ISKLMSSL)
      ```

   c. Generate an x.509v3 CERTAUTH digital certificate to RACF. The PCICC keyword is optional for the certificate authority (CA) certificate:

      ```
      RACDCERT GENCERT
      CERTAUTH
      SUBJECTSDN(CN('ISKLM CA'))
      SIZE(4096)
      NOTBEFORE( DATE( date cert_becomes_valid ))
      NOTAFTER( DATE( date_cert_expires ))
      WITHLABEL('ISKLM.CA')
      ```

   d. Generate an x.509v3 digital certificate to RACF using the PCICC keyword. Sign this certificate with the CERTAUTH certificate:

      ```
      RACDCERT GENCERT
      ID(SKLMSRV)
      SUBJECTSDN(CN('Key Encrypting Key for device based encryption'))
      SIZE(4096)
      NOTBEFORE( DATE( date cert_becomes_valid ))
      NOTAFTER( DATE( date_cert_expires ))
      WITHLABEL('Use.An.Agreed.Upon.Naming.Convention')
      SIGNWITH(CERTAUTH LABEL(‘ISKLM.CA’))
      PCICC
      ```

   e. Connect the certificates to the keyring:

      i.
      ```
      RACDCERT ID(SKLMSRV)
         CONNECT(ID(SKLMSRV)
         LABEL('Use.An.Agreed.Upon.Naming.Convention’)
         RING(ISKLMRing)
         USAGE(PERSONAL))
      ```

ii. It is also necessary to connect a CERTAUTH certificate to the keyring:

```
RACDCERT ID(SKLMSRV)
CONNECT(ID(SKLMSRV)
LABEL('ISKLM.CA')
RING(ISKLMRing)
USAGE(CERTAUTH))
```

iii. Connect a CERTAUTH certificate to the SSL keyring:

```
RACDCERT ID(SKLMSRV)
CONNECT(ID(SKLMSRV)
LABEL('ISKLM.CA')
RING(ISKLMSSL)
USAGE(CERTAUTH))
```

3. Configure IBM Security Key Lifecycle Manager to use a SAF keyring. Consider the following keyring-related configuration options:

   a. `Admin.ssl.keystore.name`:

      - This database contains the key pairs and certificates that are used for Secure Sockets 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 Layer client presents to the Secure Sockets Layer server comes from this keystore.
      - This keystore is optional but advised.
      - An example is `Admin.ssl.keystore.name = safkeyring://SKLMSRV/ISKLMSSL`.

   b. `Admin.ssl.truststore.name`:

      - This database file is used to check the trust of the Secure Sockets Layer server certificate that the server presents to the Secure Sockets Layer client.
      - This keystore is optional but advised.
      - An example is `Admin.ssl.truststore.name = safkeyring://SKLMSRV/ISKLMSSL`.

   c. `config.keystore.file`:

      - This file specifies the keystore to be used to store device-based encryption keys and certificates.
      - This keystore is required.
      - An example is `config.keystore.file = safkeyring://SKLMSRV/ISKLMRing`.

   d. `TransportListener.ssl.keystore.name`:

      - This database is used by the Security Key Lifecycle Manager for z/OS server to hold the certificate and private keys for the Secure Sockets Layer server. This certificate is given to the Secure Sockets Layer 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 Layer client.
      - This keystore is required.
      - An example is `TransportListener.ssl.keystore.name = safkeyring://SKLMSRV/ISKLMSSL`.

   e. `TransportListener.ssl.truststore.name`:

      - This database contains the public keys and signed certificates that are used to verify the identities of other clients and servers. If the property `TransportListener.ssl.clientauthentication` is not set to the default value of 0,

the Security Key Lifecycle Manager for z/OS server, acting as the Secure Sockets Layer 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 Layer client.

- This keystore is required.
- An example is `TransportListener.ssl.truststore.name = safkeyring://SKLMSRV/ISKLMSSL`.

4. The IBM Security Key Lifecycle Manager Started Task user ID must have the proper authority to key management activities:
   a. Read the keyring:

```
RDEFINE FACILITY IRR.DIGTCERT.LIST UACC(NONE)
PERMIT IRR.DIGTCERT.LIST CL(FACILITY) ID(SKLMSRV) ACCESS(READ)
```

   b. Read the keys on the keyring:

```
RDEFINE FACILITY IRR.DIGTCERT.LISTRING UACC(NONE)
PERMIT IRR.DIGTCERT.LISTRING CL(FACILITY) ID(SKLMSRV) ACCESS(READ)
```

5. The Security Administrator must have the proper authority to key management activities:
   a. Read and write to the IBM Security Key Lifecycle Manager keyrings:

```
RDEFINE RDATALIB SKLMSRV.ISKLMRing.UPD UACC(NONE)
RDEFINE RDATALIB SKLMSRV.ISKLMSSL.UPD UACC(NONE)
PERMIT SKLMSRV.ISKLMRing.UPD ID(<Security_Admin user ID>) CL(RDATALIB)
ACCESS(READ)
PERMIT SKLMSRV.ISKLMSSL.UPD ID(<Security_Admin user ID>) CL(RDATALIB)
ACCESS(READ)
```

Sysplex considerations

It is important to avoid the collisions of file writes by separate IBM Security Key Lifecycle Manager instances. In a sysplex configuration, it is possible to share z/OS UNIX System Services file systems. However, because multiple IBM Security Key Lifecycle Manager instances are not serialized, sharing common files might cause corruption and data loss.

File sharing considerations

IBM Security Key Lifecycle Manager writes to the following files:

- Keystore files
- Drivetable
- Audit metadata file
- Configuration file
- Audit log
- Debugging log

Keystore files

These files are used by the JCE to store encryption key material. IBM Security Key Lifecycle Manager uses several keystores. They might all be the same file, or various files might be used. The IBM Security Key Lifecycle Manager properties file points to the keystore files:

- `Admin.ssl.keystore.name=safkeyring\://SKLMSRV/SKLMRing`

- `TransportListener.ssl.truststore.name=safkeyring\://SKLMSRV/SKLMRing`
- `TransportListener.ssl.keystore.name=safkeyring\://SKLMSRV/SKLMRing`
- `config.keystore.file=safkeyring\://SKLMSRV/SKLMRing`
- `Admin.ssl.truststore.name=safkeyring\://SKLMSRV/SKLMRing`

Refer to the *IBM Security Key Lifecycle Manager for z/OS Version 1.1 Planning, and User's Guide,* SC14-7628-00, for detailed descriptions and usage information about these keystores.

Drivetable

This file is 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 command-line interface (CLI) client. The IBM Security Key Lifecycle Manager properties file points to this file:

```
config.drivetable.file.url=FILE\:/etc/SKLMSRV/drivetble/filedrive.table
```

Audit metadata file

The audit metadata file is `Audit.metadata.file.name=/etc/SKLMSRV/metafile.xml`.

Configuration file

The IBM Security Key Lifecycle Manager environment file, which is located through the STDENV file, points to this file:

```
export ISKLMARGS="/ftssuser/SKLMSRV/config/ISKLMConfig.properties.zos"
```

Important: IBM Security Key Lifecycle Manager for z/OS *must not be running when you edit the configuration file*. If you have previously started the Security Key Lifecycle Manager for z/OS server, you must exit it, or any changes that you make are not saved.

Audit log

The IBM Security Key Lifecycle Manager properties file points to these files:

- `Audit.handler.file.directory = /u/SKLMSRV/ISKLMCfg/logs`
- `Audit.handler.file.name = isklm_audit.log`
- `Audit.handler.class = com.ibm.ltklm.audit.file.SimpleFileSecurityEventHandler`

You can write audit records to SMF instead by setting the following property:

```
Audit.handler.class = com.ibm.ltklm.audit.smf.SMFSecurityEventHandler
```

Debugging log

The IBM Security Key Lifecycle Manager properties file points to this file:

```
debug.output.file = /u/SKLMSRV/ISKLMCfg/debug
```

Using started task JCL

The IBM Security Key Lifecycle Manager on z/OS is typically started using JCL and JzOS. The IBM JZOS Batch Toolkit for z/OS software development kit (SDK) is a set of tools that enhances Java batch capabilities and the use of system interfaces on z/OS. It includes a native launcher for running Java applications directly as batch jobs or started tasks. It also includes a toolkit of Java classes that make access to traditional z/OS data and key system services directly available from Java applications.

IBM Security Key Lifecycle Manager is a Java application that executes under z/OS UNIX Systems Services. It is started and controlled by a console wrapper that is implemented using JzOS. The console wrapper starts and stops the IBM Security Key Lifecycle Manager class. It

acts as a proxy between the z/OS operator console and the IBM Security Key Lifecycle Manager CLI. See Figure 3.

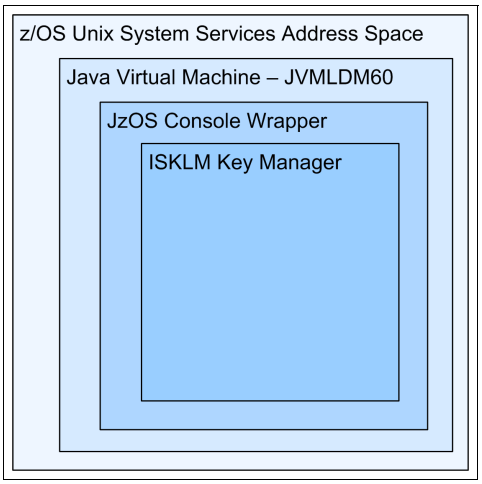

Figure 3 z/OS platform for IBM Security Key Lifecycle Manager

Sample JCL

The JOB statement in the sample JCL that is shown in Example 11 defines the following process variables. The EXEC statement of the STC JCL describes the Java Virtual Machine Load Module for the installed Java.

Example 11 Sample JCL

```
JAVACLS – this names the ISKLM Console Wrapper instantiated at startup
ARGS - Arguments to Java class (typically blank)
LIBRARY – STEPLIB dataset for JVMLDM load module
VERSION – Java version: 14, 50, 56, 60, etc.
LOGLVL – ISKLM debugging level when debug = on us configured +I(info) +T(trc)
REGSIZE – Region size required for heap and other storage (typically 0M)
LEPARM – parameters for the Language Environment (typically a null string)
```

JCL DD cards

The DD cards in Example 12 are used in the started task JCL.

Example 12 JCL DD cards

```
STEPLIB – Over-rides normal z/OS search path for load modules - optional
SYSPRINT – JOB output processing destination
SYSOUT – Unix process output processing destination
STDOUT - Unix process normal output processing destination
STDERR - Unix process error output processing destination
CEEDUMP - Unix process traceback and dump output processing destination
ABNLIGNR - shuts off Abend-AID allowing a normal IBM dump when set to DUMMY
STDENV – Location of the variable definition file for the ISKLM environment
```

The JCL in Figure 4 on page 17 represents a sample started task.

```
//ISKLM PROC JAVACLS='com.ibm.jzosekm.ISKLMConsoleWrapper' ,
//   ARGS=,                                  < Args to Java class
//   LIBRARY='SYS1.SIEALNKE',                < STEPLIB FOR JVMLDM module
//   VERSION='60',                           < 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
//* to execute the IBM 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=* denotes the default
//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(ISKLMENV),DISP=SHR
//*
```

Figure 4 JCL started task

Using the CLI as z/OS commands

The IBM Security Key Lifecycle Manager CLI provides commands to interact with the key manager. Starting IBM Security Key Lifecycle Manager as a started task using JzOS provides a console command wrapper Java class:

```
com.ibm.jzosekm.ISKLMConsoleWrapper
```

When the STC is started, it in turn starts the IBM Security Key Lifecycle Manager for z/OS key manager Java class:

```
com.ibm.ltklm.ISKLMServer
```

z/OS operator commands can be issued to the console wrapper, which will pass the command to IBM Security Key Lifecycle Manager. The console wrapper also retrieves output from the command and writes it to SYSOUT.

You issue commands to IBM Security Key Lifecycle Manager for z/OS using the z/OS MODIFY command:

```
MODIFY ISKLM,APPLID='command'
```

Figure 5 on page 17 shows the structure of the command console wrapper.

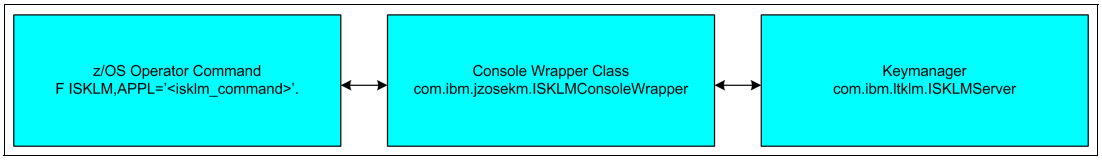

Figure 5 IBM Security Key Lifecycle Manager console wrapper

Refer to the IBM Security Key Lifecycle Manager for z/OS Information Center for a complete list of available commands:

http://publib.boulder.ibm.com/infocenter/tivihelp/v2r1/topic/com.ibm.tivoli.isklm.doc_11/top_EKMipug_cmd_line_interace.html

Sharing keys between sysplex members

To share keys between IBM Security Key Lifecycle Manager instances, it is necessary to take explicit actions based on the type of keystore in use.

Using Sysplex Distributor VIPA

The use of a Sysplex Distributor virtual IP address (VIPA) allows the installation to roll IPLs across the sysplex and not lose access to the key manager, as shown in Figure 6.

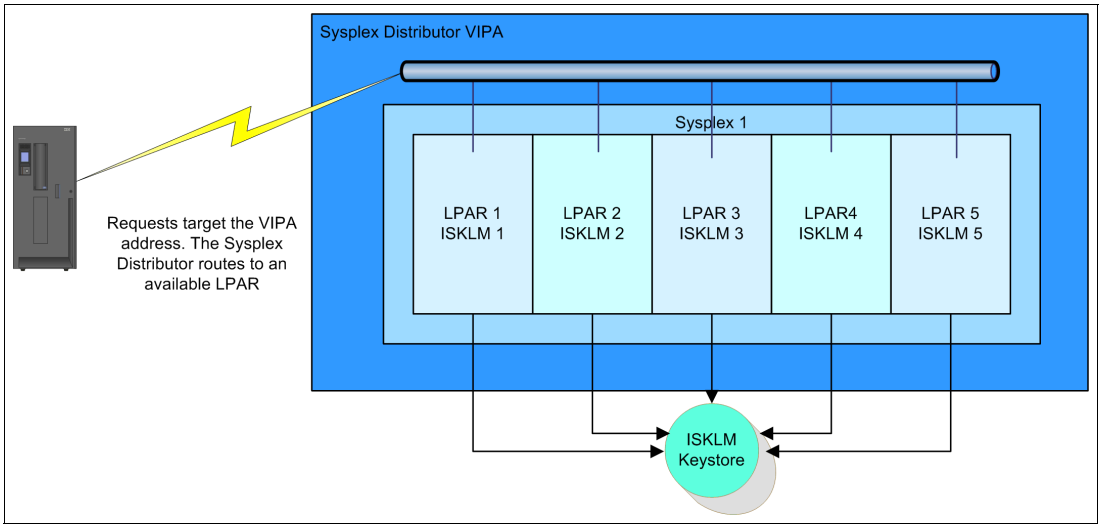

Figure 6 Sysplex Distributor VIPA

Using JCEKS

Keys that are maintained in a shared file system can be accessed by multiple members of a sysplex. The configuration of each IBM Security Key Lifecycle Manager can point to the same keystore through the `config.keystore.file` property.

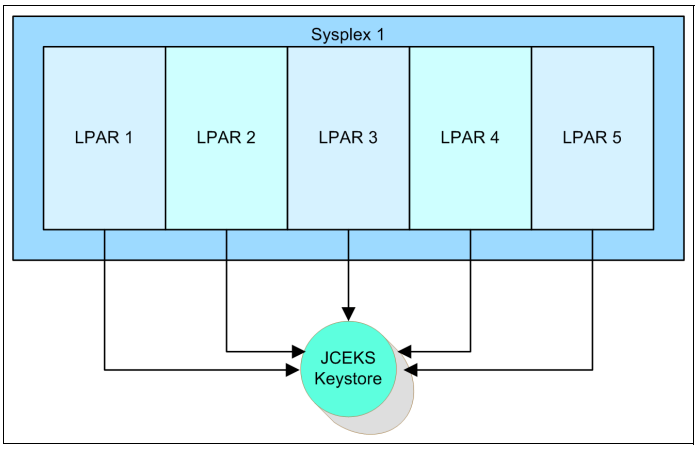

Figure 7 Sysplex sharing JCEKS

Important: Do not share other IBM Security Key Lifecycle Manager files using this method.

Using JCERACFKS

There are two methods for key sharing using JCERACFKS:

- Using the same security manager database for all logical partitions (LPARS) in the sysplex
- Using a separate security manager database for LPARS in the sysplex

Using the same security manager database for all LPARS in the sysplex

Keys that are maintained on a SAF keyring can be accessed by multiple members of a sysplex if the external security manager (RACF, ACF2, or Top Secret) is in a sysplex sharing configuration. The configuration of each IBM Security Key Lifecycle Manager can point to the same keyring through the `config.keystore.file` property. No explicit action is required to move keys between LPARs. See Figure 8.

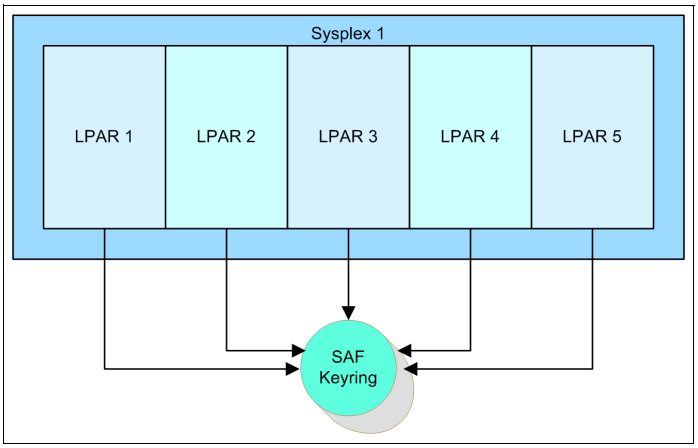

Figure 8 JCERACFKS shared in a sysplex

Using a separate security manager database for LPARS in the sysplex

Keys can be maintained on several SAF keyrings, either within or outside of the sysplex. In this case, it is necessary to export the key material from one keyring and import into the other keyring. See Figure 9 on page 20.

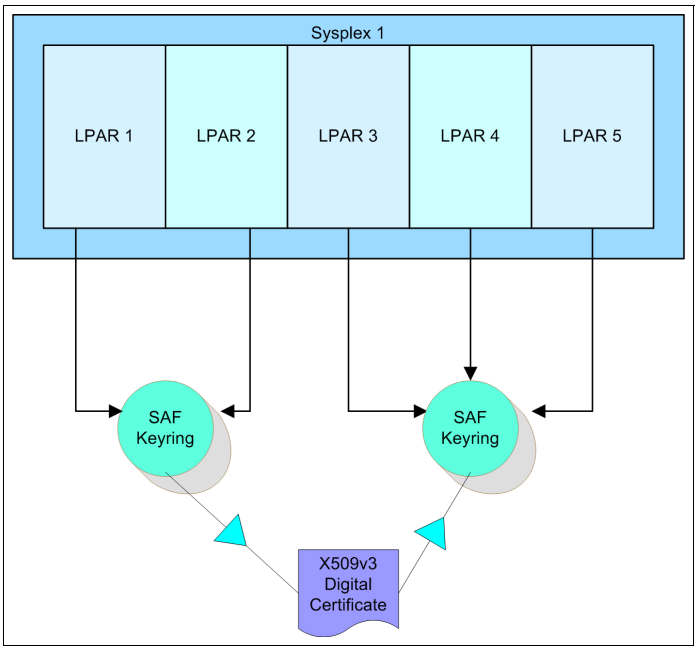

Figure 9 Multiple SAF databases in a sysplex

LTO restrictions: SAF-based keystores cannot be used to manage symmetric keys for LTO devices.

Certificates that are stored on a SAF keyring can be exported using the commands from the installed security manager. For RACF, use the RACDCERT command.

Issue the RACDCERT EXPORT command to export a digital certificate and its associated public and private keys to a password-protected sequential file. See Example 13.

Example 13 RACDCERT EXPORT command

```
RACDCERT ID(ISKLM_SERVER_ID)
EXPORT(LABEL('label-name'))
DSN('dataset_for_saved_cert')
PASSWORD(‘new_dataset_password’)
FORMAT(PKCS12DER)
```

The PKCS12DER keyword indicates to export the certificate and the private key (which must exist and must not be an ICSF or PCICC key). The package that is produced by specifying one of the PKCS #12 keywords is encrypted using the password that is specified according to the PKCS #12 standard.

Transfer the exported certificate dataset to the target z/OS image.

Add the certificate to the SAF keystore on the target image. For RACF, use the RACDCERT command. Issue the RACDCERT ADD command to add the certificate to RACF. See Example 14.

Example 14 RACDCERT ADD command

```
RACDCERT ID(ISKLM_SERVER_ID) ADD('dataset_for_saved_cert')
PASSWORD(‘new_dataset_password’) PCICC
```

Connect the certificate to the keyring. See Example 15.

Example 15 RACDCERT CONNECT command

```
RACDCERT ID(ISKLM_SERVER_ID) CONNECT(
   ID(ISKLM_SERVER_ID)
   LABEL('label_name')
   RING(ISKLMRing)
   USAGE(PERSONAL)
   )
```

Refresh the keystore cache for the active IBM Security Key Lifecycle Manager instance. Issue the REFRESHKS command at the z/OS operator console:

```
F ISKLM,APPL='REFRESHKS'
```

Using JCECCAKS

Keys maintained in a shared file system can be accessed by multiple members of a sysplex. The configuration of each IBM Security Key Lifecycle Manager can point to the same keystore through the `config.keystore.file` property. JCECCAKS uses a combination of a z/OS UNIX System Services file and the ICSF CKDS to store symmetric keys, and the ICSF PKDS to store asymmetric keys. The z/OS UNIX file, the CKDS, and the PKDS must be shared across the sysplex. See Figure 10.

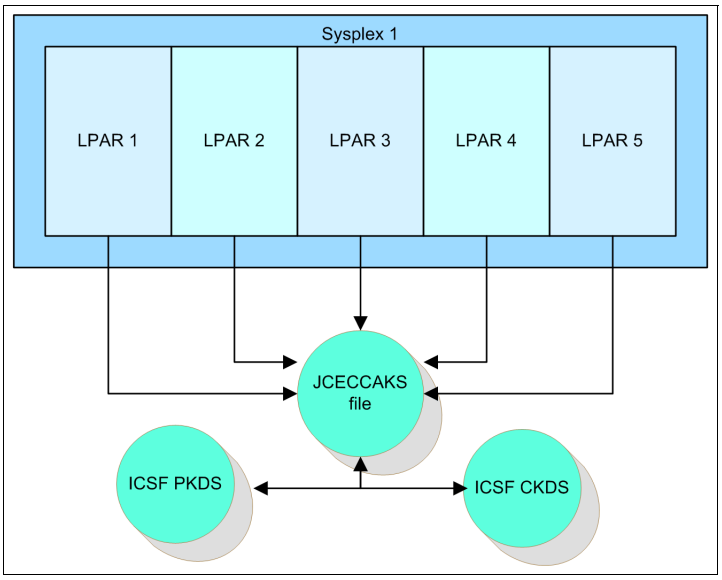

Figure 10 JCECCAKS keystore

Using JCECCARACFKS

Keys maintained on a SAF keyring can be accessed by multiple members of a sysplex if the external security manager (RACF, ACF2, or Top Secret) is in a sysplex-sharing configuration. JCECCARACFKS uses a combination of a SAF keyring and the ICSF CKDS to store asymmetric keys. The configuration of each IBM Security Key Lifecycle Manager can point to the same keyring through the `config.keystore.file` property only if the same ICSF PKDS is used. See Figure 11 on page 22.

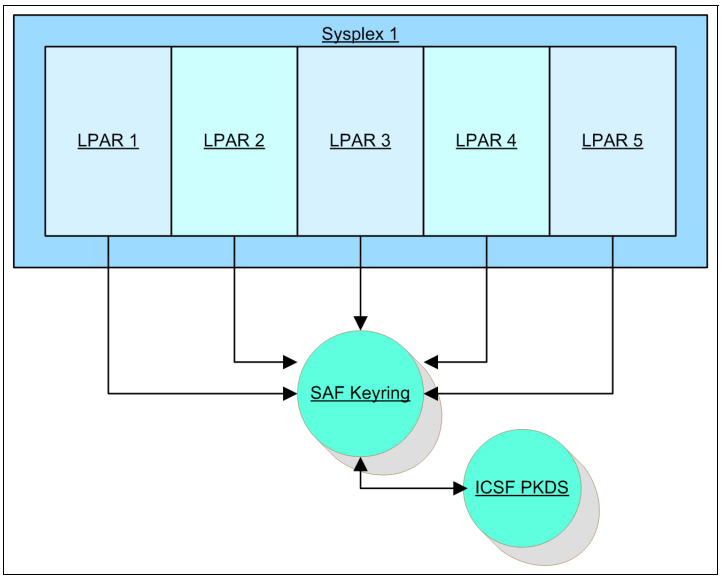

Figure 11 JCECCARACFKS shared in a sysplex

Sharing keys across multiple sysplexes

In order to share keys between IBM Security Key Lifecycle Manager instances in separate sysplexes, it is necessary to take explicit actions based on the type of keystore in use. Keys need to be exported from the keystore and imported into the keystore at the target sysplex.

Using JCEKS

When using a JCEKS keystore file, it is possible to copy the keystore file from one IBM Security Key Lifecycle Manager instance to another. In order to perform this operation cleanly, follow these steps:

1. Stop the target IBM Security Key Lifecycle Manager instance.
2. Create a backup copy of the current keystore file in use by the target IBM Security Key Lifecycle Manager instance.
3. Using FTP, Secure Copy Protocol (SCP), or another means, transfer the keystore file from the source IBM Security Key Lifecycle Manager to the target IBM Security Key Lifecycle Manager instance.
4. Rename the keystore file to match the `config.keystore.file` property at the target instance.
5. Restart the target IBM Security Key Lifecycle Manager instance.

Using JCERACFKS

SAF-based keystores cannot be used to manage symmetric keys for LTO devices.

Certificates that are stored on a SAF keyring can be exported using the commands from the installed security manager. For RACF, use the RACDCERT command.

Issue the RACDCERT EXPORT command to export a digital certificate and its associated public and private keys to a password-protected sequential file. See Example 16 on page 23.

Example 16 RACDCERT EXPORT command

```
RACDCERT EXPORT(LABEL('label-name'))
DSN('dataset_for_saved_cert')
PASSWORD(‘new_dataset_password’)
FORMAT(PKCS12DER)
```

The PKCS12DER keyword indicates to export the certificate and the private key (which must exist and must not be an ICSF or PCICC key). The package that is produced by specifying one of the PKCS #12 keywords is encrypted using the password that is specified according to the PKCS #12 standard.

Transfer the exported certificate dataset to the target z/OS image.

Add the certificate to the SAF keystore on the target image. For RACF, use the RACDCERT command. Issue the RACDCERT ADD command to add the certificate to RACF. See Example 17.

Example 17 RACDCERT ADD command

```
RACDCERT ID(ISKLM_SERVER_ID) ADD('dataset_for_saved_cert')
PASSWORD(‘new_dataset_password’) PCICC
```

Connect the certificate to the keyring. See Example 18.

Example 18 RACDCERT CONNECT command

```
RACDCERT ID(ISKLM_SERVER_ID) CONNECT(
   ID(ISKLM_SERVER_ID)
   LABEL('label_name')
   RING(ISKLMRing)
   USAGE(PERSONAL)
   )
```

Refresh the keystore cache for the active IBM Security Key Lifecycle Manager instance. Issue the REFRESHKS command at the z/OS operator:

```
F ISKLM,APPL=’REFRESHKS’
```

Using JCECCAKS

JCECCAKS uses **hwkeytool**, as depicted in Figure 10 on page 21. The methods that are used to transfer keys depends on the keytype. Keys that are stored in ICSF on z/OS are encrypted using the hardware master key. Symmetric keys are stored in the CKDS encrypted using the symmetric master key (SYM-MK). Asymmetric keys are stored in the PKDS encrypted using the asymmetric master key (ASYM-MK). See Figure 12 on page 24.

Sharing keys: This example shows sharing specific keys, but it is acceptable to share whole datasets (CKDS and PKDS) using an IDCAMS REPRO job if the master keys are the same on each system. Full *KDS sharing is the preferred method.

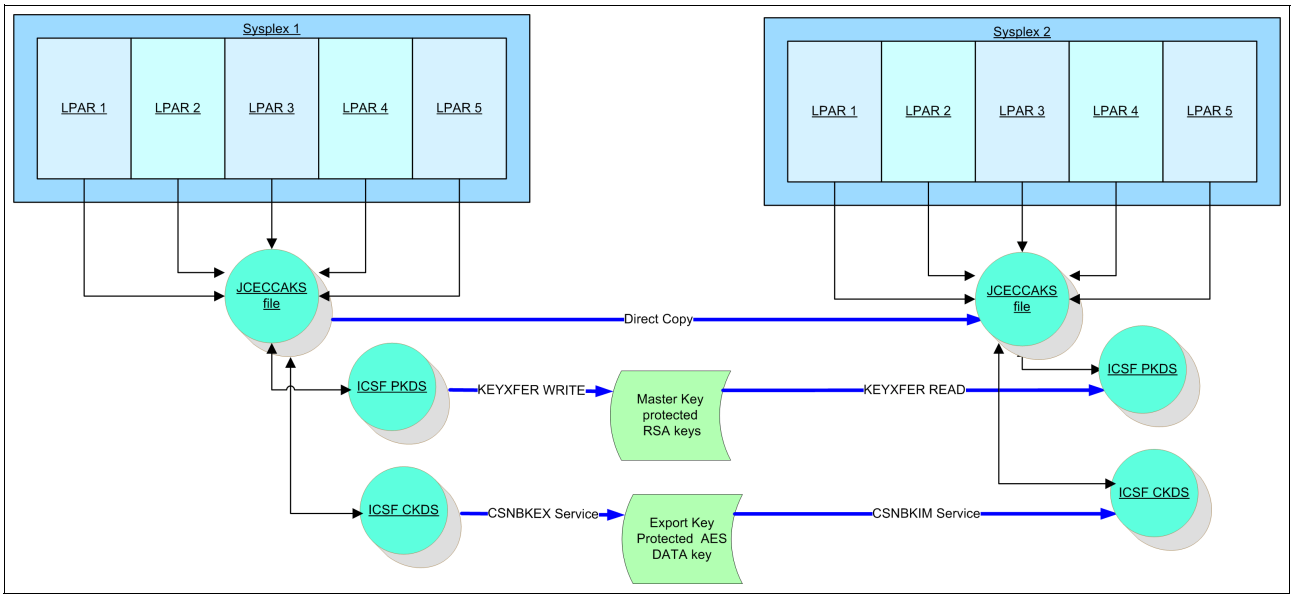

Figure 12 Sysplex sharing: JCECCAKS multiple sysplexes

Sharing symmetric keys

Symmetric keys, including Data Encryption Standard (DES), Triple DES, and AES keys, are copied from one ICSF system to another with the assistance of additional keys called transporter keys. Transporter keys are generated in ICSF using the Key Generation Utility (KGUP). They have two parts: the EXPORTER key and the IMPORTER key.

Using KGUP, create the EXPORTER key:

```
ADD TYPE(EXPORTER) CLEAR LABEL(JCECCAKSEXPORT)
```

As a result of the ADD command, the CSFKEYS dataset has an entry. This entry will become the IMPORTER key, but you first must perform several tasks with the data.

Turn HEX on in the browser. The key is in the clear as character data. We need to get the EBCDIC values. For example, a space is x'40' and the number 1 is x'F1', but they are shown vertically in HEX mode in the Interactive System Productivity Facility (ISPF) browser. See Example 19.

Example 19 ISPF browser character data

```
`O)_/.¶ZÃ._ê.@mº          <<<<<< Character data
7D5662BE61651799          <<<<<< first nibble of character
96DD1C694AD26C4B          <<<<<< last nibble of character
```

Copy this key to a string that is a flattened view of the HEX data. The translation of this example looks like the following line:

```
79D65D6D612CB6E9641A6D52167C949B
```

Split this value into two 16-digit values using a comma:

```
79D65D6D612CB6E9,641A6D52167C949B
```

Create another KGUP command to create the IMPORTER key on the target system:

```
ADD TYPE(IMPORTER) CLEAR LABEL(JCECCAKSIMPORT),
KEY(79D65D6D612CB6E9,641A6D52167C949B)
```

After it has been added, the CKDS on the source system contains this information:

```
JCECCAKSEXPORTER
EXPORTER2011060213110520.....................{.........GýR..Ý~H.þ..¼ø.. '.. ...
'............
```

And at the target system, the CKDS contains this information:

```
JCECCAKSIMPORTER
IMPORTER2011060213192384.....................{.........Åä.¢öøÿ4SÕ£3úò3Ù.â'..
...â'............
```

Refresh the in-storage CKDS on each system to ensure that the keys are available. It is now possible to move symmetric keys between the systems using ICSF Common Cryptographic Architecture (CCA) APIs.

Use the key export callable service to re-encipher any type of key (except an ANSI key-encrypting key (AKEK) or an IMPORTER key-encrypting key (IMP-PKA)) from encryption under a master key variant to encryption under the same variant of an exporter key-encrypting key. The re-enciphered key can be exported to another system.

If the key to be exported is a DATA key, the key export service generates a key token with the same key length as the input token's key. See Example 20.

Example 20 Generating a key token

```
CALL CSNBKEX(
             return_code,
             reason_code,
             exit_data_length,
             exit_data,
             key_type,
             source_key_identifier,
             exporter_key_identifier,
             target_key_identifier )
```

The source key is exported from the CKDS encrypted under the exporter key and placed in the target key. That string can then be written to a file to be transferred to another system. The associated importer key is used to decrypt the source key. The source key is then encrypted with the local SYM-MK and placed in the local CKDS. See "Sample of ICSF API usage" on page 41 for a sample REXX exec implementing this service.

Labels must be consistent across both implementations to maintain access to the key material. After the keys are copied between systems, also copy the `JCECCAKS` file. Use this `JCECCAKS` file as the value for the `config.keystore.file` property in the IBM Security Key Lifecycle Manager configuration file.

Sharing asymmetric keys

Hardware-protected asymmetric keys are shared using a two-step process:

1. Transfer the keys from the ICSF PKDS using KEYXFER.
2. Copy the JCECCAKS file using **scp** or secure FTP.

KEYXFER facilitates the transfer of public key algorithm (PKA) key tokens (RSA keys) between systems that use the Integrated Cryptographic Services Facility (ICSF).

This tool is available from the following website:

`http://www-03.ibm.com/systems/z/os/zos/features/unix/bpxa1ty2.html`

The key transfer tool (KEYXFER) is a REXX exec that runs on z/OS.

The KEYXFER tool assumes that the following conditions exist:

- ICSF runs on the systems that are involved in the key transfer.
- ICSF has an active PKDS.
- All systems that are involved in the transfer use the same ASYM master key.

The tool retrieves a PKA key token from the active PKDS and writes it to a dataset. The dataset can then be transmitted to any number of systems. On each system, the tool can be used to read the key token from the transmitted dataset and store it into the active PKDS. The tokens are referenced by PKDS label.

The command uses the following format:

```
KEYXFER OPER, PLABEL, DSN, OPTION
```

Where:

OPER	READ from the dataset or WRITE to the dataset.
PLABEL	The label of the PKDS record to be retrieved or stored.
DSN	The name of the dataset that holds the token.
OPTION	OVERWRITE a label in the PKDS. If OVERWRITE is specified in the option field, an existing PKDS label will be overwritten with the token from the input dataset.

For the dataset, a physical sequential (PS) dataset or partitioned dataset (PDS) can be used. An LRECL=80 is recommended, but it is not required.

The information that is stored in the KEYXFER dataset consists of the following information:

- Date
- PKDS label
- Length of token
- Token

Exchanging keys: Public key tokens and external private key tokens can be received on any ICSF system. If the PKA key token is an internal private key token (see *z/OS Cryptographic Services ICSF: Application Programmer's Guide,* SA22-7522-15), it is encrypted under the ICSF master key of the system. Transferring the key token requires that the receiving systems use the same ICSF master key.

If ICSF services are RACF-protected (CSFSERV), access will be required by the user for the CSNDKRC, CSNDKRR, and CSNDKRW services.

KEYXFER command examples

The following examples use KEYXFER:

- Use the following command to write the key token that is stored in the active PKDS under the label PKDS.KEY.LABEL to the dataset TEMP.MEM:

  ```
  KEYXFER WRITE, PKDS.KEY.LABEL, TEMP.MEM
  ```

- Use the following command to read the key token that is contained in the dataset TEMP.MEM and write the token to the active PKDS under the label PKDS.KEY.LABEL. If the label already exists in the PKDS, the operation will fail.

  ```
  KEYXFER READ, PKDS.KEY.LABEL, TEMP.MEM
  ```

- Use the following command to read the key token that is contained in the dataset TEMP.MEM and write the token to the active PKDS under the label PKDS.KEY.LABEL. If the label already exists in the PKDS, the token for that label will be overwritten.

  ```
  KEYXFER READ, PKDS.KEY.LABEL, TEMP.MEM, OVERWRITE
  ```

- Use the following command to read the key token that is contained in the dataset TEMP.MEM and write the token to the active PKDS. Because no PLABEL was specified, the label from the original system will be extracted from the file and used as the label for the token on the new system.

  ```
  KEYXFER READ, , TEMP.MEM
  ```

- Use the following command to copy the JCECCAKS file to the target system and ensure that the filename is correct and noted in the IBM Security Key Lifecycle Manager configuration properties file. Refresh the IBM Security Key Lifecycle Manager keystore.

  ```
  F ISKLM,APPL='REFRESHKS'
  ```

Using IDCAMS

Copy a CKDS or PKDS dataset using IDCAMS REPRO to a flat file. That flat file can be transferred to a target system and restored using IDCAMS REPRO. See Example 21.

Example 21 IDCAMS REPRO transfer and restore example

```
//FLATCKDS  JOB  …
//STEP1   EXEC PGM=IDCAMS,REGION=0M
//SYSPRINT DD  SYSOUT=*
//NEWDD  DD DSN=SYS1.CSFCKDS,DISP=SHR
//OUTDD DD DSN=USER.REPRO.FLATCKDS,
//         UNIT=3390,VOLUME=SER=TSO123,DISP=(NEW,CATLG),
//         DCB=(RECFM=FB,LRECL=252,BLKSIZE=32760),
//         SPACE=(CYL,(620,80))
//SYSIN DD *
 REPRO INFILE(NEWDD) OUTFILE(OUTDD)
```

Example 22 shows how to restore this dataset on a target system.

Example 22 IDCAMS REPRO restore example

```
//FLATCKDS  JOB  …
//STEP1   EXEC PGM=IDCAMS,REGION=0M
//SYSPRINT DD  SYSOUT=*
//OUTDD  DD DISP-=SHR,DSN=SYS1.CSFCKDS
//INDD DD DISP=SHR,DSN=USER.REPRO.FLATCKDS
//SYSIN DD *
 REPRO INFILE(INDD) OUTFILE(OUTDD)
```

Using JCECCARACFKS

JCECCARACFKS keystores store information in the SAF external security manager (RACF, ACF2, or Top Secret) and ICSF. SAF-based keystores cannot be used to manage symmetric keys for LTO devices.

Moving ICSF-based RSA key material between z/OS images requires the use of the KEYXFER tool. KEYXFER facilitates the transfer of PKA key tokens (RSA keys) between systems that use the Integrated Cryptographic Services Facility (ICSF). Keys are copied from the PKDS to a file. These keys are encrypted under the Asymmetrical Hardware Master Key (ASYM-MK) in the cryptographic module.

Both the SAF-based digital certificate and the ICSF-based RSA keys must be moved using the same label names across all images. See Figure 13.

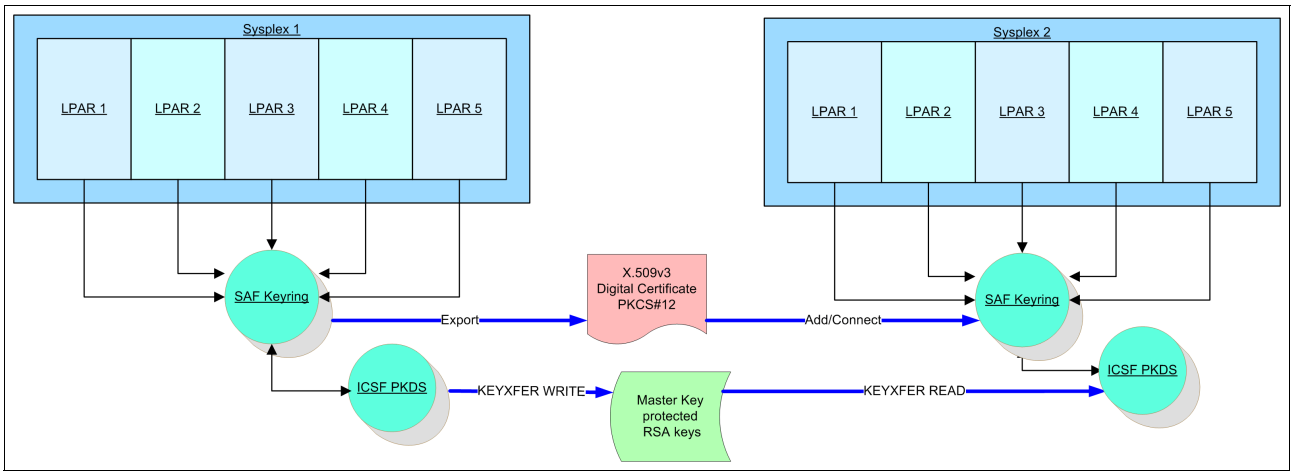

Figure 13 Sysplex Sharing: JCECCARACFKS multiple sysplexes

This tool is available from the following website:

`http://www-03.ibm.com/systems/z/os/zos/features/unix/bpxa1ty2.html`

The key transfer tool (KEYXFER) is a REXX exec that runs on IBM MVS™. The KEYXFER tool assumes the following conditions:

- ICSF runs on the systems that are involved in the key transfer.
- ICSF has an active PKA Key dataset (PKDS).
- All systems that are involved in the transfer use the same ASYM master key.

The tool retrieves a PKA key token from the active PKDS and writes it to a dataset. The dataset can then be transmitted to any number of systems. On each system, the tool can be used to read the key token from the transmitted dataset and store it into the active PKDS. The tokens are referenced by the PKDS label.

The command uses the following format:

`KEYXFER OPER, PLABEL, DSN, OPTION`

Where:

OPER	READ from the dataset or WRITE to the dataset.
PLABEL	The label of the PKDS record to be retrieved or stored.
DSN	The name of the dataset that holds the token.
OPTION	OVERWRITE a label in the PKDS. If OVERWRITE is specified in the option field, an existing PKDS label will be overwritten with the token from the input dataset.

Asymmetric keys are represented in the system by a digital certificate in the security manager and the key material in the ICSF PKDS. Use the following procedure to propagate this information:

1. Export the digital certificate from the SAF security manager to a sequential file, as shown in Example 23 on page 29.

Example 23 RACDCERT EXPORT command

```
RACDCERT EXPORT(LABEL('label-name'))
DSN('dataset_for_saved_cert')
PASSWORD(‘new_dataset_password’)
FORMAT(PKCS12DER)
```

2. Copy the RSA keys from the ICSF PKDS to a dataset using KEYXFER WRITE:

   ```
   KEYXFER WRITE, PKDS.KEY.LABEL, TRANSFER.FILE
   ```

3. Copy the newly created datasets to the target system:

   - DATASET.FOR.SAVED.CERT
   - TRANSFER.FILE

4. Copy the RSA keys from the dataset into ICSF using KEYXFER READ:

   ```
   KEYXFER READ, PKDS.KEY.LABEL, TRANSFER.FILE
   ```

5. Add the certificate to the SAF keystore on the target image. For RACF, use the RACDCERT command. Issue the RACDCERT ADD command to add the certificate to RACF:

   ```
   RACDCERT ID(ISKLM_SERVER_ID) ADD('dataset_for_saved_cert')
   PASSWORD(‘new_dataset_password’) PCICC
   ```

   PCICC keyword: When the certificate is added with the PCICC keyword, an association is made with the RSA keys that were added to ICSF with the same label using KEYXFER. Internal processes assure that the keys and the certificates are associated correctly.

6. Connect the certificate to the keyring, as shown in Example 24.

Example 24 RACDCERT CONNECT command

```
RACDCERT ID(ISKLM_SERVER_ID) CONNECT(
      ID(ISKLM_SERVER_ID)
      LABEL('label_name')
      RING(ISKLMRing)
      USAGE(PERSONAL)
      )
```

7. Refresh the keystore cache for the active IBM Security Key Lifecycle Manager instance. Issue the REFRESHKS command at the z/OS operator:

   ```
   F ISKLM,APPL='REFRESHKS'
   ```

Auditing options

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 (the directory and file name are configurable). The file size of these files is also configurable. As records are written to the file, the size of the file reaches the configured size. Then, the file is closed and renamed based on the current timestamp. Another file is then opened, and records are written to the newly created file. The overall log of audit records is separated into files of configurable size. The file names are sequenced by the timestamp of the point at which the size of the file exceeds the configurable size.

Security Key Lifecycle Manager for z/OS provides System Management Facilities (SMF) 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 subtype 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 following website:

http://publib.boulder.ibm.com/infocenter/zos/v1r12/index.jsp?topic=/com.ibm.zos.r12.icha800/toc.htm

Using SMF records

IBM Security Key Lifecycle Manager for z/OS is capable of using SMF to maintain the audit trail of logged events. Set the Audit.handler.class configuration property:

```
Audit.handler.class = com.ibm.ltklm.audit.smf.SMFSecurityEventHandler
```

SMF must be set to capture type 83 subtype 6 records. Alter the active IEASMFxx member of the SYS1.PARMLIB concatenation:

```
SYS(TYPE(83(6))
```

Event records are extracted from SMF using the SMF Data Unload Utility. The RACF SMF Data Unload Utility (IRRADU00) enables installations to create a sequential file from the security-relevant audit data. You can use the sequential file in several ways:

- View it directly
- Use it as input for installation-written programs
- Manipulate it with sort/merge utilities
- Output it to an XML-formatted file for viewing on a web browser
- Upload it to a database manager (for example, DB2) to process complex inquiries and create installation-tailored reports

The sequential file is not intended to be used directly as input to RACF commands.

Viewing audit records using DB2

Example 25 shows an example JCL to pull all type 83 subtype 6 records for SYS1.MANA. The output of this job is formatted text in the OUTDD dataset, USER1.ISKLM.SMFOUT.

Example 25 Viewing audit records with DB2

```
//SMFDUMP  EXEC PGM=IFASMFDP
//SYSPRINT DD  SYSOUT=A
//ADUPRINT DD  SYSOUT=A
//OUTDD    DD  DISP=SHR,DSN=USER1.ISKLM.SMFOUT
//SMFDATA  DD  DISP=SHR,DSN=SYS1.MANA
//SMFOUT   DD  DUMMY
//SYSIN    DD  *
     INDD(SMFDATA,OPTIONS(DUMP))
     OUTDD(SMFOUT,TYPE(83(6)))
     ABEND(NORETRY)
     USER2(IRRADU00)
     USER3(IRRADU86)
/*
```

The dataset USER1.ISKLM.SMFOUT has the attributes that are shown in Example 26.

Example 26 Dataset attributes

```
Organization  . . . : PS
Record format . . . : VB
Record length . . . : 12288
Block size  . . . . : 27998
```

The output of IRRADU00 is a set of variable-length records. This dataset must be allocated as a variable-length dataset, with a logical record length (LRECL) of at least 12288. If a shorter LRECL is supplied, IRRADU00 changes the LRECL to 12288.

IRRADU00 also changes the block size of the dataset to be at least four more than the LRECL, unless the block size was set to zero to allow the system to choose the best block size.

The records that are produced by the RACF SMF Data Unload Utility are designed to be processed by the DB2 Load Utility or its equivalent. The definition and control statements for a DB2 utilization of the output, all of which are contained in SYS1.SAMPLIB, are listed:

- Sample data definition language (DDL) statements to define the relational presentation of the audit information and sample DB2 definitions, which perform database and index creation. These statements are in member IRRADUTB.
- Sample control statements for the DB2 Load Utility that map the output from the RACF SMF Data Unload Utility. These statements are in member IRRADULD.
- Sample Structured Query Language (SQL) queries that demonstrate useful inquiries that can be made. These queries are in member IRRADUQR.

Viewing audit records using XML

Example 27 shows an example JCL to pull all type 83 subtype 6 records for SYS1.MANA. The output of this job is XML in the XMLFORM dataset, USER1.ISKLM.XMLFORM.

Example 27 Viewing audit records with XML

```
//SMFISKLM EXEC PGM=IFASMFDP
//SYSPRINT DD  SYSOUT=*
//ADUPRINT DD  SYSOUT=*
//SMFDATA  DD  DISP=SHR,DSN=SYS1.MANA
//*OUTDD    DD  DISP=SHR,DSN=USER1.PRIV.SMFOUT
//XMLFORM  DD  DISP=SHR,DSN=USER1.ISKLM.XMLFORM
//*
//SMFOUT   DD  DUMMY
//SYSIN    DD  *
  INDD(SMFDATA,OPTIONS(DUMP))
  OUTDD(SMFOUT,TYPE(83(6)))
  ABEND(RETRY)
  USER2(IRRADU00)
  USER3(IRRADU86)
//*
```

OUTDD DD statement: Note that the OUTDD DD statement is commented out when producing XML output.

The output of IRRADU00 is a set of variable-length records. This dataset must be allocated as a variable-length dataset, with a logical record length (LRECL) of at least 12288. If a shorter LRECL is supplied, IRRADU00 changes the LRECL to 12288.

IRRADU00 also changes the block size of the dataset to be at least four more than the LRECL, unless the block size was set to zero to allow the system to choose the best block size.

On z/OS, you can process the document using the IBM XML Toolkit for z/OS. The XML can be used in the following ways:

- Viewed using the ISPF edit function
- Viewed using an XML-capable web browser
- Converted to HTML using a style sheet
- Processed by an XML parser and processor

On other systems, such as personal computers and workstations, the audit report can be viewed using an XML-capable web browser. Many browsers that are available today have the ability to correctly parse and render XML documents. Therefore, after the audit report is on that system, you can read it as easily as any other web document. Simply, display a listing of the files, and single-click or double-click the file to open it in the browser window. The platform documentation can help you discover which applications are able to parse and display XML files.

Note that to use the XML file on a personal computer, you must first alter the EBCDIC encoding line at the top of the file:

```
<?xml version='1.0' encoding='ebcdic-cp-us' ?>
```

So, it looks like the following line:

```
<?xml version='1.0' encoding='ISO8859-1' ?>
```

An EBCDIC end-of-file x'01A' appears as the last byte of the XML file. Remove this byte when viewing the file on a non-EBCDIC system, such as a personal computer.

Using XML style sheets

A *style sheet* helps format the data in the XML file when it is rendered at a browser or XML editor. Example 28 shows a sample style sheet and the output from an IBM Security Key Lifecycle Manager for z/OS audit log.

Example 28 XML sample style sheet

```
<?xml version='1.0' encoding='ISO8859-1' ?>
<xsl:stylesheet version="1.0"
xmlns:rac='http://www.ibm.com/xmlns/zOS/IRRSchema'
xmlns:d="http://www.ibm.com/xmlns/zOS/TKLMSchema"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output encoding="ibm-1047"/>
<xsl:template match="/">
<html>
<body>
<h2>Event List</h2>
<table border="1">
<tr bgcolor="#AAAAAAA">
<th align="left">Date</th>
<th align="left">Time</th>
<th align="left">Result</th>
```

```
<th align="left">event Data</th>
</tr>
<xsl:for-each select="rac:securityEventLog/rac:event">
<tr>
<td><xsl:value-of select="rac:dateWritten"/></td>
<td><xsl:value-of select="rac:timeWritten"/></td>
<td><xsl:value-of select="rac:eventQual"/></td>
<td width="500" align="left" valign="top"><xsl:value-of
select="rac:details/d:eventData"/></td>
</tr>
</xsl:for-each>
</table>
</body>
</html>
</xsl:template>
</xsl:stylesheet>
```

IBM Security Key Lifecycle Manager for z/OS event list

Table 3 shows a sample event list for IBM Security Key Lifecycle Manager for z/OS.

Table 3 IBM Security Key Lifecycle Manager for z/OS event list

Date	Time	Result	Event data
2011-06-13	12:48:36.55	SUCCESS	Runtime event:Ý… timestamp=Mon Jun 13 12:48:36 EDT 2011… ComponentId=ÝthreadId=ThreadÝmain,5,main¨¨… event source=com.ibm.ltklm.ISKLMServer… outcome=Ýresult=successful¨… event type=SECURITY_RUNTIME… resource=Ýname=ISKLMAdmin;type=application¨… action=runISKLMServer… user=Ýname=ISKLMAdmin¨… ¨…
2011-06-13	12:48:37.12	SUCCESS	Resource management event:Ý… timestamp=Mon Jun 13 12:48:37 EDT 2011… ComponentId=ÝthreadId=ThreadÝmain,5,main¨¨… event source=com.ibm.ltklm.keygroups.KeyGroupManager… outcome=Ýresult=successful¨… event type=SECURITY_MGMT_RESOURCE… action=retrieve… user=Ýname=KMSAdmin¨… resource=Ýname=safkeyring://SKLMSRV/SKLMRing;type=file¨… ¨…
2011-06-13	12:48:37.29	SUCCESS	Resource management event:Ý… timestamp=Mon Jun 13 12:48:37 EDT 2011… ComponentId=ÝthreadId=ThreadÝmain,5,main¨¨… event source=com.ibm.ltklm.keystore.KeyStoreLoader… outcome=Ýresult=successful¨… event type=SECURITY_MGMT_RESOURCE… action=retrieve… user=Ýname=KMSAdmin¨… resource=Ýname=safkeyring://SKLMSRV/SKLMRing;type=file¨… ¨…

Date	Time	Result	Event data
2011-06-13	12:48:37.34	SUCCESS	Resource management event:Ý… timestamp=Mon Jun 13 12:48:37 EDT 2011… ComponentId=ÝthreadId=ThreadÝmain,5,main¨¨… event source=com.ibm.ltklm.keystore.KeyStoreLoader… outcome=Ýresult=successful¨… event type=SECURITY_MGMT_RESOURCE… action=retrieve… user=Ýname=KMSAdmin¨… resource=Ýname=safkeyring://SKLMSRV/SKLMSSL;type=file¨… ¨…
2011-06-13	2011-06-13	SUCCESS	Runtime event:Ý… timestamp=Mon Jun 13 12:48:37 EDT 2011… ComponentId=ÝthreadId=ThreadÝmain,5,main¨¨… event source=com.ibm.ltklm.ISKLMServer… outcome=Ýresult=unsuccessful¨… event type=SECURITY_RUNTIME… message=no symmetric Key aliases LTO drives not supported. ErrorCode= 19… resource=Ýname=if LTO support is needed valid symmetric Keys must be added to the config keystore;type=file¨… action=stop… ¨…

Audit log file

IBM Security Key Lifecycle Manager for z/OS can maintain the audit trail of logged events in a z/OS UNIX System Services text file. Set the Audit.handler.class configuration property:

```
Audit.handler.class = com.ibm.ltklm.audit.file.SimpleFileSecurityEventHandler
```

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, the size of the file reaches the configurable size. Then, the file is 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 files, whose size is configurable. Their names are sequenced by the timestamp of the point at which the size of the file exceeds the configurable size.

Example 29 shows a typical audit record that is written in this format.

Example 29 Audit log file

```
Resource management event:Ý
  timestamp=Sat Jun 11 12:45:45 EDT 2011
  ComponentId=ÝthreadId=ThreadÝmain,5,main¨¨
  event source=com.ibm.ltklm.keystore.KeyStoreLoader
  outcome=Ýresult=successful¨
  event type=SECURITY_MGMT_RESOURCE
  action=retrieve
  user=Ýname=KMSAdmin¨
  resource=Ýname=safkeyring://SKLMSRV/SKLMRing;type=file¨
```

z/OS conversion: The normal z/OS EBCDIC code page converts square brackets: `[ = Ý and ] = ¨`

Troubleshooting on z/OS

Most problems concerning the Security Key Lifecycle Manager for z/OS involve configuration or starting the server.

Successful startup

Example 30 shows a successful server startup.

Example 30 Successful startup

```
BPXMO23I (SKLMSRV) Loaded drive key store successfully
BPXMO23I (SKLMSRV) Loading admin keystore...
BPXMO23I (SKLMSRV) No symmetric keys in symmetricKeySet, LTO drives
cannot be supported.
BPXMO23I (SKLMSRV) Starting the Security Key Lifecycle Manager
1.1-20110222
BPXMO23I (SKLMSRV) Processing Arguments
BPXMO23I (SKLMSRV) Contact IBM support at 1-800-IBM-SERV (1-800-426-7378
) or through your normal business channel.
BPXMO23I (SKLMSRV) Processing
BPXMO23I (SKLMSRV) Server is started
BPXMO23I (SKLMSRV) Server is running. TCP port: 3801, SSL port: 443
```

Error: Default keystore failed to load

This error is common when first starting IBM Security Key Lifecycle Manager on z/OS. Example 31 shows the trace-back in the STC log.

Example 31 Default keystore failed to load error

```
com.ibm.ltklm.KeyManagerException: Default keystore failed to load
 at
com.ibm.ltklm.keygroups.KeyGroupManager.loadDefaultKeyStore(KeyGroupManager.java:1
58)
 at com.ibm.ltklm.keygroups.KeyGroupManager.init(KeyGroupManager.java:314)
 at com.ibm.ltklm.ISKLMServer.c(ISKLMServer.java:271)
 at com.ibm.ltklm.ISKLMServer.<init>(ISKLMServer.java:282)
 at com.ibm.ltklm.ISKLMServer.a(ISKLMServer.java:530)
 at com.ibm.ltklm.ISKLMServer.main(ISKLMServer.java:238)
 at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
 at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:48)
 at
sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:
25)
 at java.lang.reflect.Method.invoke(Method.java:600)
 at com.ibm.jzosekm.ISKLMConsoleWrapper.a(ISKLMConsoleWrapper.java:25)
 at com.ibm.jzosekm.ISKLMConsoleWrapper.main(ISKLMConsoleWrapper.java:35)
```

Several possible causes for this error exist:

- The keystore does not exist.

  This situation usually occurs when the keystore name is incorrect in the properties file. Ensure that the keystore file or keyring name is correct on all properties. Also, ensure that the user ID is correct if using a SAF keyring. The user ID is case sensitive.

- The STC user does not have authority to the keystore.

  To read a SAF keyring, the user ID needs authority to the IRR.DIGTCERT.LIST and IRR.DIGTCERT.LISTRING profiles in the FACILITY class.

- The keystore was tampered with or the password was incorrect.

  This situation usually occurs during setup when changing keystore types. IBM Security Key Lifecycle Manager will obfuscate the password in the properties file. To reset the obfuscated password, change the property from `config.keystore.password.obfuscated=07089E88A9A9AD9DA08B` to `config.keystore.password=<password of keystore>`.

- A configuration mismatch exists between the defined services providers and the types of keystores:
  - If any of the keystores that are described in the IBM Security Key Lifecycle Manager properties file are of a separate type, this error occurs. Ensure that all keystores are declared as the same type, and that the type is correct.
  - If using JCECCAKS, JCERACFKS, or JCECCARACFKS, ensure that the correct cryptographic services provider is declared in `java.security`.

- An incorrect value is set in the `STDENV` file for the IJO variable.

  If a mismatch exists between the keystore type and the IJO variable, IBM Security Key Lifecycle Manager will fail to load the keystore.

Errors:

- This error can appear if any of the configured keystores or truststores are in error.
- Always check the `isklm_audit.log` file for further information regarding start-up failures.

Error: NoClassDefFoundError

Example 32 shows the NoClassDefFoundError.

Example 32 NoClassDefFoundError

```
-> StaticMethod.invoke()
com.ibm.jzosekm.ISKLMConsoleWrapper.main
Could not find or load class: com.ibm.jzosekm.ISKLMConsoleWrapper
-> JniUtil.writeStackTrace()
JVMJZBL2007E Stack trace follows:
java.lang.NoClassDefFoundError: com.ibm.jzosekm.ISKLMConsoleWrapper
Caused by: java.lang.ClassNotFoundException: com.ibm.jzosekm.ISKLMConsoleWrapper
 at java.net.URLClassLoader.findClass(URLClassLoader.java:423)
 at java.lang.ClassLoader.loadClass(ClassLoader.java:653)
 at sun.misc.Launcher$AppClassLoader.loadClass(Launcher.java:346)
 at java.lang.ClassLoader.loadClass(ClassLoader.java:619)
JVMJZBL2999T <- JniUtil.writeStackTrace()
```

This error indicates that the `com.ibm.jzosekm.ISKLMConsoleWrapper` was not found in the Java CLASSPATH or in `java/lib/ext`. This class, along with all the Java classes that are required by IBM Security Key Lifecycle Manager for z/OS, is contained in the `IBMSKLM.jar` file that was delivered as part of the SMP/E for z/OS installation.

Migration from IBM Encryption Key Manager

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, perform these steps:

1. Use the **`refresh`** command to refresh Encryption Key Manager.
2. Use the **`stopekm`** command to stop Encryption Key Manager to ensure that there is no data loss. Encryption Key Manager cannot be active during the migration.
3. Install Security Key Lifecycle Manager for z/OS on the same computer as Encryption Key Manager.
4. 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.
5. Copy and store the keystore, including all keys and certificates that the configuration file references.
6. Copy the configuration file, device table, and metadata file.
7. Copy the keygroups 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`

RACF keystores: If the keystores in use are RACF-based (JCERACFKS or JCERACFCCAKS), you do not have to back up any of the files that are referenced in the listed configuration parameters that have a keystore or truststore. You still must back up the `config.keygroup.xml.file`, `config.drivetable.file.url`, and `Audit.metadata.file.name` parameters.

Migration procedure

Use the following procedure to begin the migration:

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.

     DS8000 specifics: 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. If the Encryption Key Manager used JZOS, Security Key Lifecycle Manager for z/OS can also use the JZOS launcher to restart.
5. Check the audit log file to ensure that the migration was successful and that no errors are logged. Example 33 shows an example.

Example 33 Sample audit log file

```
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]
  ]
```

SMF addition: 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.

Common practices

Tape encryption and key management have been around for many years. The following list contains several of the practices that are common to most implementations.

Key management common practices

The following list describes the common practices that you must remember when working with key management:

1. Key backup procedures.

   A set of procedures must be in place to back up an copy of all keys in the keystore:

   – JCEKS: Store a copy of the keystore file.

   – JCERACFKS: Back up the RACF, ACF2, or Top Secret database.

   – JCECCAKS: Back up the ICSF datasets.

   – JCECCARACFKS: Back up the RACF, ACF2, or Top Secret database *and* the ICSF datasets.

2. Periodic generation and rotation of default keys and certificates:

   – Key material, whether it is symmetric AES keys for LTO devices, or asymmetric RSA keypairs stored in x.509v3 digital certificates for 3592 devices, must be generated periodically rather than using a single key or certificate indefinitely.

   – Keys need to be generated on a yearly basis, at least.

3. Controlled key dispersal:

   – Key material must be treated as controlled and sensitive data.

4. Minimum RSA keysize of 2048:

   – Key length equates to key strength. A length of 2048 must be considered a minimum.

5. Manageable keygroup size:

   – Symmetric keygroups larger than 500 can affect performance.

   – Multiple keygroups mitigate the risk of compromise.

6. Separation of duties:

   – Separate the roles of those individuals controlling the key manager processes from those individuals managing the key material.

   – Users need to be grouped, and groups must be given authority to specific tasks.

7. Archival of audit logs:

   – SMF uses generation data group (GDG).

   – Roll-off of z/OS UNIX System Services files.

8. Naming conventions must be in place for all keys, groups, and certificates:

   – Certificate label-naming standards must contain this information:

     • Ownership

     • Usage

     • Environment information

     • Active date range

9. Use as many IBM Security Key Lifecycle Managers as you can assign from the proxy, as permitted by the local security guidelines:
   - IBM Security Key Lifecycle Manager instances on multiple LPARs within a sysplex mitigate the risk of a single point of failure.
10. Copies of the keystore and configuration files must be on physically separate storage systems:
    - z/OS UNIX System Services files must be on separate file systems from the installation files:
      - Audit logs, debug files, configuration file, and metadata files must be separate from the OS files.
11. Limit access to the IBM Security Key Lifecycle Manager password.
12. Limit access to the IBM Security Key Lifecycle Manager keystore.
13. Separate the `config.keystore` from the `ssl.keystores` to avoid certificate expiration problems when synchronizing.
14. Keep your TCP/IP paths as physically separate as possible.
15. Manage your JCEKS passwords properly, by using split passwords and long strong passwords.
16. Keep the IBM Security Key Lifecycle Manager Java separate from the system Java to remove update issues, preferably within the IBM Security Key Lifecycle Manager ID file structure.
17. Create a separate ID under which the IBM Security Key Lifecycle Manager will run, preferably neither UID 0 nor Administrator.
18. Keep a copy of the Java environment, with the IBM Security Key Lifecycle Manager configured, on a CD or on a separate physical storage medium.
19. Develop a procedure that describes how to create keys and how to implement them into production.
20. Develop criteria describing the type of events that cause keys to be changed, for example, time, personnel loss, personnel responsibility change, key compromise, or cartridge loss.
21. For TS1120 technology, develop a procedure to rekey cartridges (not available for TS7700).
22. For LTO4, define a limit to the number of cartridges that any particular key can encrypt. Estimate the number of cartridges that are consumed for a period and divide that number out to determine how large the keygroup must be.

Sample of ICSF API usage

Example 34 shows a snippet of REXX code to export a data key under a key-encrypting exporter key. This code is provided as a sample and is not intended to function as is.

Example 34 Sample REXX code for exporting a data key

```
/*********************************************************/
/* ExportDataKey                                         */
/*  Use the data key export callable service to          */
/*  reencipher a data-encrypting                         */
/*  key (key type of DATA only) from                     */
/*  encryption under the master key to                   */
```

```
/*  encryption under an exporter key-encrypting key.      */
/*  The reenciphered key is                               */
/*  in a form suitable for export to another system.      */
/*********************************************************/
ExportDataKey
Key_label = “label.of.the.key.to.export”
Exp_key = “exporter.key.label”

label   = LEFT(key_label,  64)
exp_key = LEFT(exp_key,  64)
full_key = copies(' ', 64)
retcx  = '00000000'x
reascx = '00000000'x
ret_cx  = '00000000'x
reas_cx = '00000000'x
exit_lenx = '00000000'x;
exit_data = "NONE    " ;
key_type = 'DATA    ':
 say 'Exporting key at label ' label ' using ' exp_key
 address linkpgm 'CSNBKEX' ,
                 'retcx'    'reascx'  ,
                 'exit_lenx' 'exit_data',
                 'key_type',
                 'label'  'exp_key',
                 'full_key';

/* at this point “full_key” contains the exported and encrypted key */


Plsace the key to be imported into a variable called Source_Key

Import the key using the CSNBKIM service :
ImportDataKey:
Source_Key = “the key being imported”
Exp_key = “importer.key.label”
Target_label = “label.of.the.key.being.imported” /* where to put the key */
label   = LEFT(target_label,  64)  /* justify all labels to 64 bytes */
exp_key = LEFT(exp_key,  64)

retcx  = '00000000'x
reascx = '00000000'x
ret_cx  = '00000000'x
reas_cx = '00000000'x
exit_lenx = '00000000'x;
exit_data = "NONE    " ;
key_type = 'DATA    ':



say 'Importing key at label ' label ' using ' exp_key
 address linkpgm 'CSNBKEX' ,
                 'retcx'    'reascx'  ,
                 'exit_lenx' 'exit_data',
                 'key_type',
                 'source_Key',
```

```
    'exp_key',
                'label'
```

The team who wrote this paper

This paper was produced by a team of specialists from around the world working at the International Technical Support Organization, Austin Center.

Axel Buecker is a Certified Consulting Software IT Specialist at the ITSO, Austin Center. He writes extensively and teaches IBM classes worldwide about areas of software security architecture and network computing technologies. He has a degree in Computer Science from the University of Bremen, Germany. He has 25 years of experience in a variety of areas related to workstation and systems management, network computing, and e-business solutions. Before joining the ITSO in March 2000, Axel worked for IBM in Germany as a Senior IT Specialist in Software Security Architecture.

William C. Johnston is adept at bringing best practices surrounding enterprise security to client processes. He is experienced in working with large system installations to deploy encryption key management solutions and has performed enterprise system security assessments. A large focus of his background is educating client teams on security-related topics. For over a decade, he was responsible for the design and implementation of the test approach definitions for security-related elements of the z/OS operating system, including their interaction with other components, the base OS, and other platforms, such as Linux and Microsoft Windows. In the past, he has performed code development, functional-level and system-level testing, and project management duties.

Thanks to the following people for their contributions to this project:

Ken Rogers, Jason G. Katonica, Jonathan M. Barney, and John Peck
IBM

Stephen J. Smith
International Technical Support Organization, Raleigh Center

Now you can become a published author, too!

Here's an opportunity to spotlight your skills, grow your career, and become a published author—all at the same time! Join an ITSO residency project and help write a book in your area of expertise, while honing your experience using leading-edge technologies. Your efforts will help to increase product acceptance and customer satisfaction, as you expand your network of technical contacts and relationships. Residencies run from two to six weeks in length, and you can participate either in person or as a remote resident working from your home base.

Find out more about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Stay connected to IBM Redbooks

- Find us on Facebook:

http://www.facebook.com/IBMRedbooks

- Follow us on Twitter:

  http://twitter.com/ibmredbooks

- Look for us on LinkedIn:

  http://www.linkedin.com/groups?home=&gid=2130806

- Explore new IBM Redbooks® publications, residencies, and workshops with the IBM Redbooks weekly newsletter:

  https://www.redbooks.ibm.com/Redbooks.nsf/subscribe?OpenForm

- Stay current on recent Redbooks publications with RSS Feeds:

  http://www.redbooks.ibm.com/rss.html

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.

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

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

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

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

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

COPYRIGHT LICENSE:

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

This document REDP-4646-01 was created or updated on October 7, 2011.

Send us your comments in one of the following ways:
- Use the online **Contact us** review Redbooks form found at:
  ibm.com/redbooks
- Send your comments in an email to:
  redbooks@us.ibm.com
- Mail your comments to:
  IBM Corporation, International Technical Support Organization
  Dept. HYTD Mail Station P099
  2455 South Road
  Poughkeepsie, NY 12601-5400 U.S.A.

Trademarks