IBM Security Key Lifecycle Manager Offering for Tape and DASD Encryption

System z Platform Evaluation Test (zPET) implemented and tested the latest tape encryption offering named IBM® Security Key Lifecycle Manager (ISKLM). We had previously installed and used the Enterprise Key Manager (EKM) offering and have now migrated to the latest offering which also supports key serving to IBM disk storage devices.

With this encryption solution, data to be encrypted is sent to the drive in the clear (as it is today in a non-encrypted form) with encryption and decryption of the data taking place outboard at the drive. ISKLM is a Java™ based application. This program has been designed to run on many different operating system platforms and is able to use different types of keystores. The four supported keystore types for z/OS® are:

- JCEKS
- JCE4758KS
- JCE4758RACFKS
- JCERACFKS

In our testing within zPET, we configured ISKLM and tested the JCE4758RACFKS (Hardware Secure Key) keystore. As for Java levels, JDK 5.0 SR5 or JDK 6.0 are the minimum levels for running ISKLM.

The following steps describe how we implemented ISKLM and migrated from EKM.

Preparing the file systems

We first created the zFS file systems and mount points (directories) that are needed for the JDK download, ISKLM product code, ISKLM logs, and ISKLM configuration files. Since our Parallel Sysplex® takes advantage of shared HFS in z/OS UNIX System Services, all the file systems are shared. We decided to have a separate directory for the configuration file instead of using the individual system `/etc` directory. We also needed a file system for the audit and debug log records. We used system names for directories within the `/KLMETC` and `/KLMLOGS` directories, such as `/KLMETC/Z1`. The size of the file system for the `/KLMETC` directory only needs to be 2 cylinders. We then created the `/ISKLM` directory and mounted a new zFS file system. We defined the JDK file system and mounted it at directory `/java/java16uk58682isklm`. We then created a file system for the ISKLM error and debug logs and mounted it at `/KLMLOGS`. We then updated our BPXPRM00 member and added the new file system mounts.

Note that with ISKLM, you can now have the audit records written to SMF instead of being file-based, which is what EKM used. We did use the SMF function; however, we decided to stay with the file-based method, which is an easier method to view the audit records. Our decision to use the file-based method requires monitoring and management of the file system size. The new SMF parameter is described later in section, “Creating the ISKLM configuration file”.

Be aware that the file size will grow and a method must be in place to ensure it does not grow too large. As tapes are encrypted, an audit record is written to the log. Also, when the ISKLM task starts and stops, records are also written to the audit log. Our volume of tape encryption is not that large, so we chose to monitor and control this manually.

Installing the Java JDK and ISKLM software

The next step was to install the JDK 6.0. We used the following **`pax`** command to uncompress the code:

```
pax -ppx -rvzf UK58682.PAX.Z
```

You can verify that the command was successful by issuing a Java **`version`** command**:**

```
export PATH=/java/java6316UK58682isklm/J6.0/bin:$PATH
java -version
     java version "1.6.0"
     Java(TM) SE Runtime Environment (build pmz3160sr8fp1-
     20100624_01(SR8 FP1))
     IBM J9 VM (build 2.4, JRE 1.6.0 IBM J9 2.4 z/OS s390-31
     jvmmz3160sr8ifx-20100609_59383 (JIT enabled, AOT enabled)
     J9VM - 20100609_059383
     JIT  - r9_20100401_15339ifx2
     GC   - 20100308_AA)
     JCL  - 20100624_01
```

Even though you can share the file system with other java applications, we decided that we would have our ISKLM use its own mounted JDK file system. This ensures that we do not risk affecting any other applications when customizing the JDK.

We copied the ISKLM code from our build site to the new `/ISKLM` directory. The directory now contains the following executable .jar file as well as samples that come with the product:

`IBMSKLM.jar` (ISKLM product jar file)
```
samples/ISKLMConfig.properties.zos
samples/PROCLIB.ISKLM
samples/PROCLIB.ISKLMENV
samples/SMF2LOG.JCL
```

Creating the user ID and defining the started task

A user ID must be created and defined to use z/OS UNIX System Services. This ID will be used by ISKLM when running as a started task. It also will be used when creating certificates in the keystore. We chose to use **klmserv** as the user ID, which we defined to RACF® using the **`adduser`** command with OMVS and TSO segments. We used a non-zero UID. Just remember that **klmserv** needs to own the configuration, debug, and error files or else permission access failures will be encountered after starting ISKLM.

Next, we prepared for a started task. We chose to use ISKLM as the started task name, which would use the RACF authority for **klmserv**. The following two RACF commands were issued by an administrator who has the SPECIAL attribute. When the task named ISKLM is started, then the **klmserv** RACF user ID will be assigned to that task**.**

```
RDEFINE STARTED ISKLM*.* STDATA(USER(KLMSERV) GROUP(SYS1) TRACE(YES))
SETROPTS RACLIST(STARTED) GENERIC(STARTED) REFRESH
```

Next we created a new member in proclib for the ISKLM started task containing the following JCL:

```
//ISKLM PROC JAVACLS='com.ibm.jzosekm.ISKLMConsoleWrapper',
//   ARGS=,                              < ARGS TO JAVA CLASS
//   LIBRARY='JZOS.JAVA6031.LOADLIB',    < 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
//*  SPECIFICALLY, TO EXECUTE THE ENTERPRISE KEY MANAGER UNDER JZOS
//*
//*********************************************************************
//ISKLM  EXEC PGM=JVMLDM&VERSION,REGION=&REGSIZE,
//   PARM='&LEPARM/&LOGLVL &JAVACLS &ARGS'
//STEPLIB  DD DSN=&LIBRARY,DISP=SHR
//SYSPRINT DD SYSOUT=*          < SYSTEM STDOUT
//SYSOUT   DD SYSOUT=*          < SYSTEM STDERR
//STDOUT   DD SYSOUT=*          < JAVA SYSTEM.OUT
//STDERR   DD SYSOUT=*          < JAVA SYSTEM.ERR
//CEEDUMP  DD SYSOUT=*
//ABNLIGNR DD DUMMY
//********************************************************************
//*  THE FOLLOWING MEMBER CONTAINS THE JVM ENVIRONMENT SCRIPT
//********************************************************************
//STDENV DD DSN=KLMSERV.ENCRYPT.CONFIG(&SYSNAME.),DISP=SHR
//*
```

JVM environment script

In this step, we created the partitioned data set (PDS) named `KLMSERV.ENCRYPT.CONFIG`, which was specified in the started task JCL above. We created a system-named member in this data set for each of our systems that will be running ISKLM. This allows the use of a single started task member that directs the task to the system-unique configuration settings. This is accomplished via the system symbolic, **&SYSNAME**, used in the started task JCL above.

The following is an example of what we used in our `KLMSERV.ENCRYPT.CONFIG(Z1)` data set. Note this configuration parameter in the example below which specifies system Z1:

```
export ZZZZ="/KLMETC/Z1/ISKLMConfig.properties.zos"
```

The following example shows the contents of the data set:

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

#Set these variables to the installation unique values
# ISKLM_HOME = directory where ISKLM runs from
# JAVA_HOME = directory where Java is mounted
export ISKLM_HOME="/ISKLM"
export JAVA_HOME=/java/java16uk58682isklm/J6.0

export PATH=/bin:"${JAVA_HOME}"/bin:

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

LIBPATH="$LIBPATH":"${JAVA_HOME}"/lib/s390
LIBPATH="$LIBPATH":"${JAVA_HOME}"/lib/s390/j9vm
LIBPATH="$LIBPATH":"${JAVA_HOME}"/bin/classic

export LIBPATH="$LIBPATH":

# Customize your CLASSPATH here
#CLASSPATH=${JAVA_HOME}/lib
CLASSPATH="${JAVA_HOME}"/lib:"${JAVA_HOME}"/lib/ext
#CLASSPATH=$CLASSPATH:/usr/lpp/ISKLM/IBMSKLM.jar
CLASSPATH=$CLASSPATH:/ISKLM/IBMSKLM.jar

export CLASSPATH="$CLASSPATH":
#export ZZZZ="/usr/lpp/ISKLM/ISKLMConfig.properties.zos"
export ZZZZ="/KLMETC/Z1/ISKLMConfig.properties.zos"
export XXXX="com.ibm.ltklm.ISKLMServer"
export JZOS_MAIN_ARGS="$XXXX $ZZZZ"
```

```
# Configure JVM options (if any)
IJO="-Djava.protocol.handler.pkgs=com.ibm.crypto.hdwrCCA.provider"

export IBM_JAVA_OPTIONS="$IJO "
```

Copying unrestricted policy files

Another task that we had to do during the setup was to replace the `US_export_policy.jar` and local `policy.jar` files in our `$JAVA_HOME/lib/security` directory with unrestricted versions of these files. These unrestricted policy files are required by ISKLM in order to serve AES keys. The preferred method to do this on z/OS is to copy the unrestricted policy files that are shipped in the z/OS Java SDK build under the jce demo directory into the `$JAVA_HOME/lib/security` directory. If we did not do this, we would have found that we can encrypt a tape successfully, but then a decrypt from a tape would fail.

We copied these two files into the `$JAVA_HOME/lib/security` directory, as shown in this example:

```
/$JAVA_HOME/J6.0/demo/jce/policy-files
       cp unrestricted/* /java/java16uk58682isklm/J6.0/lib/security
```

Add hardware provider #2 to the Java security table

Another customization that we had to make was to add the hardware provider to the Java security table. The line shown below needs to be inserted as the second provider, and then the remaining lines need to have the provider number changed in ascending order.

Note that this is only needed if using ICSF for hardware-based keys. We added the following line to the Java security table located in the java file in the `$JAVA_HOME/lib/security` directory:

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

The following is a portion of how the Java security table looks after customization:

```
/java/java16uk58682isklm/J6.0/lib/security/java
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
security.provider.7=com.ibm.xml.crypto.IBMXMLCryptoProvider
security.provider.8=com.ibm.xml.enc.IBMXMLEncProvider
```

Creating the ISKLM configuration file

The next task we performed was to create the configuration file that ISKLM needs. Similar to how we had EKM configured, we placed the file under the system-specific directory name within `/KLMETC`. This example shows the JCE4758RACFKS keystore that we use. The configuration file is called `ISKLMConfig.properties.zos` and it is located in the `/KLMETC/Z1` directory (where `Z1` is the system on which we are running). The configuration file is customized for a specific system and should not be shared. The configuration file for ISKLM is almost the same as what was used for EKM so if you are migrating from the prior EKM product, we suggest comparing the EKM configuration file to this new file and just copying the values.

There are two new parameters in the configuration file that were introduced with ISKLM. One new parameter is for DASD key serving. This parameter is named **`ds8k.acceptUnknownDrives`**. The other new parameter is for utilizing SMF for audit record reporting, instead of file-based recording. This is the configuration parameter if you choose to use SMF for this:

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

We also suggest reviewing *IBM Security Key Lifecycle Manager for z/OS Planning and User's Guide*, available in the IBM Security Key Lifecycle Manager Information Center at http://publib.boulder.ibm.com/infocenter/tivihelp/v2r1/index.jsp?topic=/com.ibm.tivoli.isklm.doc_11/ic-homepage.html, for specifics concerning each property's settings.

One item to note is that you do not want to update the properties table once the ISKLM program is running, as changes made to the file will be lost when the server is shut down. ISKLM writes back to this file upon shutdown and will update the settings that may have been made dynamically when it was running. Only make updates to this configuration file while the server is not active. Also note that you will need the initial password you had used for this keystore when you ran EKM. Once you have started ISKLM, the configuration file will automatically be updated and the password field will be obfuscated so the password is not visible.

This is an example of the contents of our ISKLM configuration file located at `/KLMETC/Z1/ISKLMConfig.properties.zos`:

```
Audit.metadata.file.cachecount = 0
config.keystore.password.obfuscated = 5E08F5DF000004F4F7E2
TransportListener.ssl.port = 1443
config.keygroup.xml.file = FILE:/KLMETC/Z1/keygroups.xml
TransportListener.tcp.port = 3801
TransportListener.tcp.timeout = 0
TransportListener.ssl.keystore.password.obfuscated = B1084832535357474A35
fips = Off
Admin.ssl.keystore.name = safkeyring://TAPEKMS/KeyStore4758
config.keystore.provider = IBMJCECCA
```

```
TransportListener.ssl.clientauthentication = 0
TransportListener.ssl.ciphersuites = JSSE_ALL
Audit.handler.file.size = 10000
zOSCompatibility = false
drive.acceptUnknownDrives = true
Audit.metadata.file.name = /KLMETC/Z1/metadata/SKLMData.xml
TransportListener.ssl.truststore.name = safkeyring://TAPEKMS/KeyStore4758
Audit.handler.file.directory = /KLMLOGS/Z1/logs
TransportListener.ssl.protocols = SSL_TLS
TransportListener.ssl.host = localhost
Admin.ssl.keystore.password.obfuscated = 0C08A38DAEAEB2A2A590
debug.output = simple_file
config.keystore.file = safkeyring://TAPEKMS/KeyStore4758
TransportListener.ssl.keystore.name = safkeyring://TAPEKMS/KeyStore4758
Audit.eventQueue.max = 0
debug.output.file = /KLMLOGS/Z1/debug
Audit.handler.file.name = isklm_audit.log
config.keystore.type = JCECCARACFKS
requireHardwareProtectionForSymmetricKeys = true
drive.default.alias2 = Tape_Sol_Tst_Shr_Pvt_2048_Lbl_03
drive.default.alias1 = Tape_Sol_Tst_Shr_Pvt_1024_Lbl_01
Audit.event.outcome = success,failure
debug = all
Audit.event.types = all
ds8k.acceptUnknownDrives = true
Admin.ssl.truststore.name = safkeyring://TAPEKMS/KeyStore4758
config.drivetable.file.url = FILE:/KLMETC/Z1/filedrive.table
```

RACF updates

Issue the following RACF commands to define the RACF resources and permissions that allow you to create and manage the certificates. If you already have these classes defined, then skip down to the commands that issue the permits. We already had these classes defined so we only had to issue the permits, as noted below.

```
RDEFINE FACILITY IRR.DIGTCERT.ADD UACC(NONE)
RDEFINE FACILITY IRR.DIGTCERT.ADDRING UACC(NONE)
RDEFINE FACILITY IRR.DIGTCERT.DELRING UACC(NONE)
RDEFINE FACILITY IRR.DIGTCERT.LISTRING UACC(NONE)
RDEFINE FACILITY IRR.DIGTCERT.CONNECT UACC(NONE)
RDEFINE FACILITY IRR.DIGTCERT.REMOVE UACC(NONE)
RDEFINE FACILITY IRR.DIGTCERT.LIST UACC(NONE)
RDEFINE FACILITY IRR.DIGTCERT.ALTER UACC(NONE)
RDEFINE FACILITY IRR.DIGTCERT.DELETE UACC(NONE)
RDEFINE FACILITY IRR.DIGTCERT.GENCERT UACC(NONE)
RDEFINE FACILITY IRR.DIGTCERT.GENREQ UACC(NONE)
RDEFINE FACILITY IRR.DIGTCERT.EXPORT UACC(NONE)
RDEFINE FACILITY IRR.DIGTCERT.EXPORTKEY UACC(NONE)
RDEFINE FACILITY IRR.DIGTCERT.MAP UACC(NONE)
RDEFINE FACILITY IRR.DIGTCERT.ALTMAP UACC(NONE)
RDEFINE FACILITY IRR.DIGTCERT.DELMAP UACC(NONE)
RDEFINE FACILITY IRR.DIGTCERT.LISTMAP UACC(NONE)
RDEFINE FACILITY IRR.DIGTCERT.ROLLOVER UACC(NONE)
RDEFINE FACILITY IRR.DIGTCERT.REKEY UACC(NONE)


PERMIT IRR.DIGTCERT.ADD CLASS(FACILITY) ID(KLMSERV) ACCESS(CONTROL)
PERMIT IRR.DIGTCERT.ALTER CLASS(FACILITY) ID(KLMSERV) ACCESS(CONTROL)
PERMIT IRR.DIGTCERT.DELETE CLASS(FACILITY) ID(KLMSERV) ACCESS(CONTROL)
PERMIT IRR.DIGTCERT.LIST CLASS(FACILITY) ID(KLMSERV) ACCESS(CONTROL)
PERMIT IRR.DIGTCERT.ADDRING CLASS(FACILITY) ID(KLMSERV) ACCESS(CONTROL)
PERMIT IRR.DIGTCERT.DELRING CLASS(FACILITY) ID(KLMSERV) ACCESS(CONTROL)
PERMIT IRR.DIGTCERT.LISTRING CLASS(FACILITY) ID(KLMSERV) ACCESS(CONTROL)
PERMIT IRR.DIGTCERT.CONNECT CLASS(FACILITY) ID(KLMSERV) ACCESS(CONTROL)
PERMIT IRR.DIGTCERT.REMOVE CLASS(FACILITY) ID(KLMSERV) ACCESS(CONTROL)
PERMIT IRR.DIGTCERT.MAP CLASS(FACILITY) ID(KLMSERV) ACCESS(CONTROL)
PERMIT IRR.DIGTCERT.LISTMAP CLASS(FACILITY) ID(KLMSERV) ACCESS(CONTROL)
PERMIT IRR.DIGTCERT.ALTMAP CLASS(FACILITY) ID(KLMSERV) ACCESS(CONTROL)
PERMIT IRR.DIGTCERT.DELMAP CLASS(FACILITY) ID(KLMSERV) ACCESS(CONTROL)
PERMIT IRR.DIGTCERT.REKEY CLASS(FACILITY) ID(KLMSERV) ACCESS(CONTROL)
PERMIT IRR.DIGTCERT.ROLLOVER CLASS(FACILITY) ID(KLMSERV) ACCESS(CONTROL)
```

Setting up TCP/IP for ISKLM

To ensure that key requests would always be obtainable when requested from either one of our systems in our test environment, we chose to configure TCP/IP for higher availability by using Dynamic VIPA and Sysplex Distributor.

Configuring the ISKLM TCP/IP profile

For all images that are running ISKLM, the following port definitions are required in the TCP/IP profile.

Ports 3801 and 1443 must be reserved in the **`PORT`** section

```
PORT
     1443 TCP ISKLM                 ; Key Manager SSL
     3801 TCP ISKLM                 ; Key Manager
```

If the port is already being used, the **`SHAREPORT`** option must be added to each of the applications using the port. We already had port 1443 in use for IMWEB. This is an example of the setting we used:

```
PORT
     1443 TCP ISKLM SHAREPORT       ; Key Manager
     1443 TCP IMWEB SHAREPORT       ; Web server
```

SYSPLEX Distributor and Dynamic VIPA configuration

We configured a Sysplex Distributor using distributed DVIPAs for higher availability.

In zPET, we have a four-way sysplex with ISKLM running on three of the images.
The Sysplex Distributor distributes each key request to an image based on a configurable distribution method, such as WLM, ROUNDROBIN, and so on. In the case of a stack failure, the Sysplex Distributor will move to another image where the requests will be handled. The backup stack is configured in the TCP/IP profile. If one of the images to which the key requests are being sent fails, the Sysplex Distributor will stop sending requests to that image until it is back up.

Sysplex Distributor setup and Dynamic VIPA configuration is well documented in the following z/OS Communications Server publications:

z/OS Communications Server IP Configuration Reference, SC31-8776
z/OS Communications Server IP Configuration Guide, SC31-8775

A few steps are needed to get a Sysplex Distributor set up:

1. Dynamic XCF is needed. We used the following path for the requests thru the sysplex distributor:

   ```
   IPCONFIG DYNAMICXCF 192.168.49.30 255.255.255.0 3
   ```

   We set this up on all images to which the Sysplex Distributor will be distributing or that will be a backup to the Sysplex Distributor.

2. We defined the required DVIPA parameters in the **`VIPADYNAMIC`** section of the profile, as follows:

   ```
   VIPADYNAMIC
   Sysplex Distributor configuration.

   VIPADEFINE MOVEABLE IMMED 255.255.255.0 9.xx.xx.xxx
   VIPADISTRIBUTE DEFINE SYSPLEXP DISTM ROUNDROBIN 9.xx.xx.xxx PORT
   3801 1443
   DESTIP ALL
   ENDVIPADYNAMIC
   ```

   This is saying that all requests that come to DVIPA address 9.*xx.xx.xxx* on port 3801 or port 1443 are to be sent to `ALL` images in the sysplex that are configured to accept requests (Dynamic XCF setup).

3. For the images that we configure as a backup, we needed Dynamic XCF, ISKLM running. The following section tells the sysplex what DVIPAs we are backing up:

```
VIPADYNAMIC
VIPABACKUP     10   9.xx.xx.xxx
ENDVIPADYNAMIC
```

Setting up shared keys in ICSF

-
We mainly used the JCE4758RACFKS keystore in our testing. We performed the following steps to set up our ICSF environment using RACF. We had a set of predefined shared keys which were created by another test group. Below are the steps we took to setup ICSF to use the ISKLM with a JCECCARACFKS keystore.

For additional information, see *IBM Encryption Key Manager Component for the Java Platform Introduction, Planning, and User's Guide,* GA76-0418.

Note: For this ISKLM installation, since we are using the same keyring that we had previously created for EKM, we did not have to issue these commands again. The command examples shown below are what we used when we first installed EKM. At that time, we used TAPEKMS as the user ID under which the started task ran. Since we are now running the started task under the KLMSERV user ID, we only had to issue permits to `TAPEKMS.KEYSTORE4758.LST` in the RDATALIB class and ICSF resources in the CSFSERV class.

1. Configured ICSF to use a shared PKDS called `SYS1.PKDSPLX2` across all systems in our sysplex. (See *z/OS Cryptographic Services Integrated Cryptographic Service Facility System Programmer's Guide,* SA22-7520*,* for information about setting up a shared PKDS.)

2. Created the keyring:

   ```
   RACDCERT ADDRING(KeyStore4758) ID(TAPEKMS)
   ```

3. We issued appropriate access to the keyring function RACF profiles:

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

4. Generated a self-signed certificate authority certificate:

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

5. Generated two certificate/RSA key pairs for the ISKLM server instance on z/OS:

```
RACDCERT ID(TAPEKMS) GENCERT SUBJECTSDN(CN('ITOperations')
O('MyCo') C('US')) WITHLABEL('Tape_Sol_Tst_Shr_Pvt_1024_Lbl_20')
PCICC(ITOPS.KLM2.CERT)
SIZE(1024) SIGNWITH(CERTAUTH LABEL('MyLocalRACFCA'))

RACDCERT ID(TAPEKMS) GENCERT SUBJECTSDN(CN('ITOperations2')
O('MyCo') C('US')) WITHLABEL('Tape_Sol_Tst_Shr_Pvt_1024_Lbl_21')
PCICC(ITOPS.KLM1.CERT)
SIZE(1024) SIGNWITH(CERTAUTH LABEL('MyLocalRACFCA'))
```

6. Refreshed the RACF profiles:

```
SETROPTS RACLIST(DIGTCERT) REFRESH
```

7. To ensure the ISKLM server certificates and designated certificate authority are connected to the keyring of the ISKLM, we issued the following commands:

```
RACDCERT ID(TAPEKMS) CONNECT(CERTAUTH
LABEL('MyLocalRACFCA') RING(KeyStore4758))

RACDCERT ID(TAPEKMS)
CONNECT(LABEL('Tape_Sol_Tst_Shr_Pvt_1024_Lbl_20')
RING(KeyStore4758))

RACDCERT ID(TAPEKMS)
CONNECT(LABEL('Tape_Sol_Tst_Shr_Pvt_1024_Lbl_21')
RING(KeyStore4758))
```

8. Gave the ISKLM server instance RACF authority to the key label of the private keys stored in the ICSF PKDS:

```
RDEFINE CSFKEYS ITOPS.KLM1.CERT UACC(NONE)
PERMIT ITOPS.KLM1.CERT CLASS(CSFKEYS) ID(TAPEKMS)
ACCESS(READ)

RDEFINE CSFKEYS ITOPS.KLM2.CERT UACC(NONE)
PERMIT ITOPS.KLM2.CERT CLASS(CSFKEYS) ID(TAPEKMS)
ACCESS(READ)
```

Setting up the DFSMS environment for tape key serving

This section shows how we had set up our DFSMS environment when we first installed EKM. There were no changes to this when we migrated to ISKLM.

```
Data Class
   FILTLIST TAPE_UNITS INCLUDE('3590','3490','3590-1','3590-H','3592-2E','3592-3E')
Storage Class
   FILTLIST TAPE_LIST INCLUDE('3590','3490','3590-1','3592-2E','3592-3E')
Storage Group
   WHEN ('SCLSAT3') SET &STORGRP = 'SGRPAT3'
```

Adding IOS support for the TS1120 tape drive

ISKLM uses the same IOS settings that were implemented for EKM. If you are installing tape encryption for the first time, then you will need to perform similar updates, shown below, to your system's parmlib.

We added the **EKM** parameter in the IECIOS*xx* parmlib member in support of the encryption-capable TS1120 tape drive. In order to use in-band key management, we modified our IECIOS*xx* member to include the TCP/IP-related information needed to direct the IOS proxy to an appropriate key manager (primary and secondary).

For each EKM, we specified the host name with the port as follows in the IECIOS*xx* parmlib member:

```
EKM PRIMARY=9.xx.xx.xxx:3801
EKM SECONDARY=9.xx.xx.xxx:3801
```

You can use the following command to display the host names for the primary and secondary EKM.

```
D IOS,EKM
```

We also used the `SETIOS EKM` command to dynamically change the settings for the EKM, as follows:

```
SETIOS,EKM PRIMARY=9.xx.xx.xxx:3801
SETIOS,EKM SECONDARY=9.xx.xx.xxx:3801
```

We used the following sample JCL to encrypt the data on a 3992 Cartridge:

```
//ENCRYPT JOB
,,MSGLEVEL=(1,1),CLASS=A,MSGCLASS=R,NOTIFY=&SYSUID,
//    REGION=0M
/*JOBPARM SYSAFF=*
//*
//STEP2    EXEC PGM=IEBGENER
//SYSUT1   DD DSN=PET.ENCRYPTION.TEST,DISP=SHR,
//            UNIT=3390
```

```
//SYSUT2    DD UNIT=CRYPTAPE,DSN=PET.ENCY.TAPE,LABEL=(1,SL),
//          DATACLAS=ENCRYPT,DISP=OLD,VOL=SER=XXXXXX
//SYSPRINT DD SYSOUT=*
//SYSIN    DD DUMMY
//*
```

We used the following JCL to decrypt the data from Tape:

```
//DECRYPT JOB ,,MSGLEVEL=(1,1),CLASS=A,MSGCLASS=R,NOTIFY=&SYSUID,
//    REGION=0M
/*JOBPARM SYSAFF=*
//*
//STEP2    EXEC PGM=IEBGENER
//SYSUT1   DD UNIT=CRYPTAPE,DSN=PET.ENCY.TAPE,LABEL=(1,SL),
//            DATACLAS=ENCRYPT,DISP=OLD,VOL=SER=XXXXXX
//SYSUT2   DD DSN=ENCRYPT.DUMP.DATA,DISP=(NEW,CATLG),VOL=SER=XXXXXX,
//            UNIT=3390,SPACE=(TRK,(1000,0))
//SYSPRINT DD SYSOUT=*
//SYSIN    DD DUMMY
//*
```

Migration from EKM

As mentioned earlier, we previously had been running EKM and now will be running ISKLM. From a migration perspective, these are the migration steps we needed to do.

We copied the following EKM files to the new ISKLM directory:

- Configuration file (`/KLMETC/Z1/ISKLMConfig.properties.zos`)
- Device table (`/KLMETC/Z1/filedrive.table`)
- Metadata file (`/KLMETC/Z1/metadata/SKLMData.xml`)

Since we are running ISKLM under a new started task name, we had to issue RACF permits to the RDATALIB, FACILITY, CSFSERV and CSFKEYS classes that previously were defined under the EKM user ID. Following are the commands we issued:

```
pe IRR.DIGTCERT.LISTRING class(facility) ID(klmserv) acc(read)
pe IRR.DIGTCERT.LIST class(facility) ID(klmserv) acc(read)
pe TAPEKMS.KEYSTORE4758.LST CL(RDATALIB)id(klmserv) acc(read)
pe CSF-PKDS-DEFAULT CL(CSFKEYS) id(klmserv) acc(read)
pe CSFDSG CL(CSFSERV) id(klmserv) acc(read)
pe CSFDSV CL(CSFSERV) id(klmserv) acc(read)
pe CSFKGN CL(CSFSERV) id(klmserv) acc(read)
pe CSFOWH CL(CSFSERV) id(klmserv) acc(read)
pe CSFPKG CL(CSFSERV) id(klmserv) acc(read)
pe CSFPKI CL(CSFSERV) id(klmserv) acc(read)
```

```
      pe CSFPKX CL(CSFSERV) id(klmserv) acc(read)
      pe CSFRNGL CL(CSFSERV) id(klmserv) acc(read)
      pe CSFSYI CL(CSFSERV) id(klmserv) acc(read)
      pe CSFSYX CL(CSFSERV) id(klmserv) acc(read)
```

Note: We reviewed the ISKLM sample configuration file and customized accordingly using the EKM configuration file as a guide.

Running ISKLM and operational aspects

After we completed the installation and customizations, the operational aspects were quite easy. Since it is defined as a started task, all we had to do was issue the **`S ISKLM`** command on each system on which it was configured to run. Since we are using an ICSF hardware keystore, we made sure the ICSF task was running before ISKLM was started. If we did not do this, the server would not be able to connect to the keystore and ISKLM startup would fail.

Following are the syslog messages we encountered when ISKLM started:

```
18.40.21 S0042797  IEF403I ISKLM - STARTED - TIME=18.40.21
18.40.25 S0042797  BPXM023I (KLMSERV)  337
   337             ISKLM wrapper V2.1 for ISKLM1.1
   337             ISKLM console interaction is now available.
   337             To submit commands to the ISKLM from the console:
   337                 F ISKLM,APPL='ISKLM command'
   337             To stop the ISKLM properly:
   337                 P ISKLM
18.40.34 S0042797  BPXM023I (KLMSERV) Loaded drive key store successfully
18.40.34 S0042797  BPXM023I (KLMSERV) Starting the Security Key Lifecycle
Manager 1.1-20110126
18.40.34 S0042797  BPXM023I (KLMSERV) Processing Arguments
18.40.34 S0042797  BPXM023I (KLMSERV) Contact IBM support at 1-800-IBM-
SERV (1-8 00-426-7378) or through your normal business channel.
18.40.34 S0042797  BPXM023I (KLMSERV) Processing
18.40.34 S0042797  BPXM023I (KLMSERV) Server is started
18.40.35 S0042797  BPXM023I (KLMSERV) Server is running. TCP port: 3801,
SSL port : 1443
```

In the syslog messages above, notice the following message, which indicates the code build level of ISKLM:

```
Starting the Security Key Lifecycle Manager 1.1-20110126,
```

Also, the following message indicates that the ISKLM server is connected to TCP/IP and is listening on ports 3801 and 1443:

```
BPXM023I (KLMSERV) Server is running. TCP port: 3801, SSL port : 1443
```

Shutting down the ISKLM is also easy. When we wanted to stop ISKLM, we issued the **`P ISKLM`** command, and the server stopped cleanly.

The ISKLM help command, **`F ISKLM,APPL='HELP'`**, can be issued to provide a list of valid ISKLM commands.

For details about the various commands, see the ISKLM reference materials.

Additional information on ISKLM can be found in the IBM Security Key Lifecycle Manager for z/OS Information Center at:
http://publib.boulder.ibm.com/infocenter/tivihelp/v2r1/index.jsp?topic=/com.ibm.tivoli.isklm.doc_11/ic-homepage.html