Dataset Encryption

Transporting AES encrypted data keys from one z/OS host to another

This document can be found on the web, www.ibm.com/support/techdocs
Search for document number WP##### under the category of "White Papers.

Version Date: November 25, 2017

IBM Systems LBS France

Philippe RICHARD
Consulting IT specialist
Philippe_richard@fr.ibm.com

Special Notices

This document is the result of work and experiments which were done during a Redbook residency about Dataset Encryption. It was produced by the members who participated in this residency, and reviewed by Eysha S. Powers, Enterprise Cryptography, IBM Systems.
This document is presented "As-Is" and IBM does not assume responsibility for the statements expressed herein. It reflects the opinions of the authors of this redpaper.
If you have questions about the contents of this document, please direct them to Philippe Richard LBS France (philippe_richard@fr.ibm.com).

Trademarks

Table of Contents

Introduction, Notes about this document and/or Acknowledgments 4

Transporting keys 5

Starting point configurations 6

Scenario 1: 6

Site A and B have the same master key, same key label 6

Scenario 2: 15

Site A and B have different master keys, same key label 15

Scenario 3: 34

Changing the key label scenario 35

Introduction, Notes about this document and/or Acknowledgments

The purpose of this document is to describe the procedures that can be used to transport secured AES encryption data keys form one host system to another.

This paper was written during a redbook residency, and will be available in a future redbook publication once it becomes published (01/2018).

The authors of this article are:

- Thomas Liu, z/OS Systems Programmer working at Australia and New Zealand Banking Group Limited in Australia ANZ (Thomas.Liu@anz.com)
- Philippe Richard, IBM WW curriculum lead developer for z/OS, LBS, IBM France (philippe_richard@fr.ibm.com)

Thank you to the following reviewers:

Eysha S. Powers, Enterprise Cryptography,IBM Systems

Transporting keys

Dataset Encryption is one of the major pillars of Pervasive Encryption.

When you start implementing dataset encryption, you will start asking yourself:

- How can i exchange my encrypted data with my other LPARs or remote site?
- How can i exchange my encrypted data with my business partners?
- Should i send my data along with its encryption key?
- How can i send my secured keys?

The purpose of this document is to describe the procedures that can be used to transport encryption keys form one host system to another.

We will cover different scenarios that we have implemented, and for each we will describe the procedures that we have used to perform the key transportation.

Here are the starting point configurations:

We have 2 sites A and B, which need to exchange/transport encrypted datasets and keys.

__ 1. Site A and B have the same master key, and use different key labels, i.e the key KeyA that SITEA wants to send to SITEB is currently not used or defined in SITEB. This would be a typical configuration for a customer having separate environments (like production, development / qualification,…), that need to exchange encrypted datasets: what would be the recommended procedure(s) to perform the export/import..

__ 2. Site A and B have different master keys, and currently use different key labels, i.e the key KeyA that SITEA wants to send to SITEB is currently not used or defined in SITEB. This would be typically the configuration when you have 2 business partners trying to exchange encrypted datasets, and therefore exchange their encryption keys.

__ 3. Site A and B have the same master key or different master key, and the key with the key label KeyA they want to exchange is already currently defined and used in both sites; in that case what would be the process to copy/export the encrypted key from site A to site B. Should i import the key under a different key label ?

Scenario 1

Site A and B have the same master key, and use different key labels , i.e the key KeyA that SITEA wants to send to SITEB is not currently used or defined in SITEB.

Below is a chart that depicts that configuration in our context:

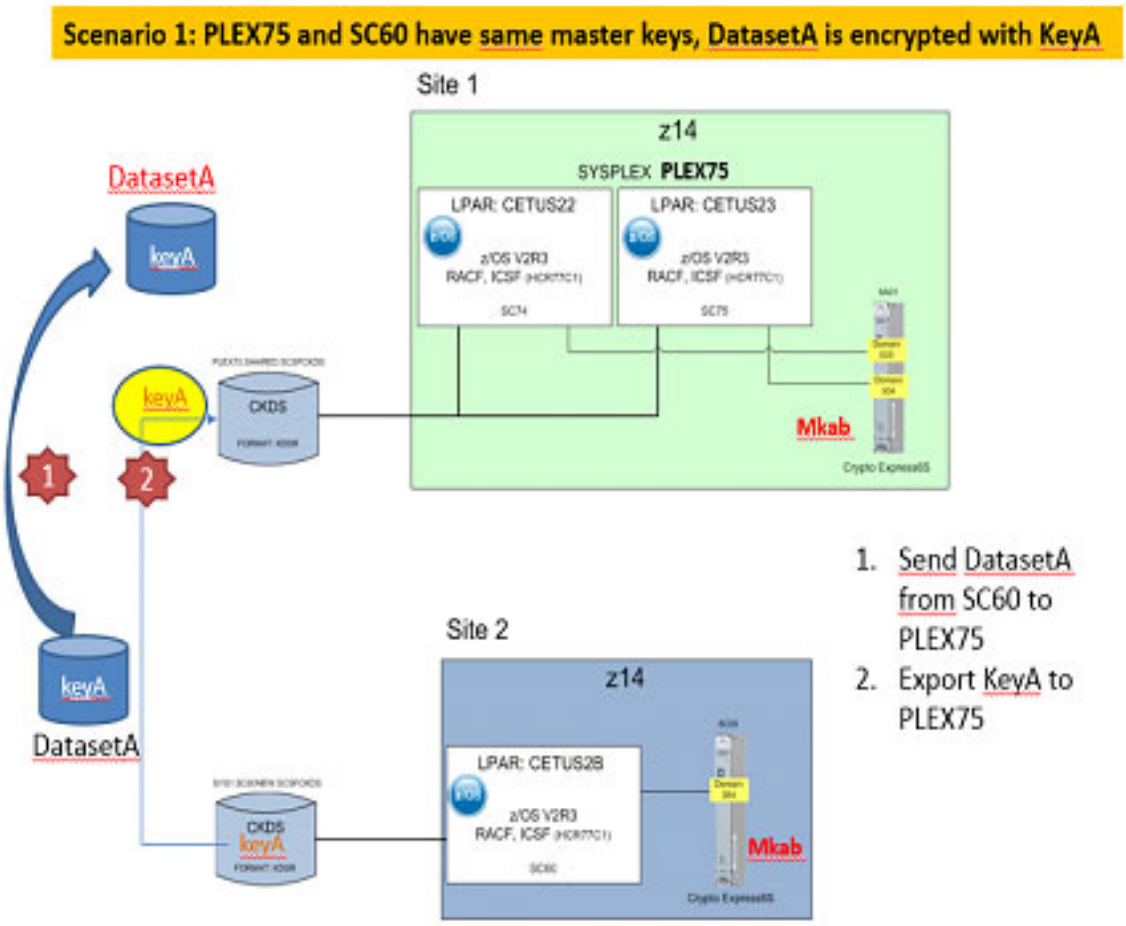

Figure 2-1. Transporting AES DATA keys between LPARs or sites that have same MKs

This is easiest and most straightforward configuration, as it allows you to use a standard z/OS utility to perform the key exchange via a REPRO.

Here is the procedure we followed:

__ a. repro 1 key to a SEQ dataset in site A

__ b. send the SEQ file to site B

__ c. on site B repro back in the key from the SEQ file into the CKDS

__ d. then refresh

__ e. check that ICSF is happy and can list this new key label ;-

__ f. define the profile in CSFKEYS for this new key label

__ g. export encrypted dataset from site A to site B (dump site A /restore in site B)

__h. try to access encrypted dataset in site B

Below are the details on our procedure:

__ a. repro 1 key to a SEQ dataset in site A

Option 1:

You can repro one or more key using IDCAMS, using a variation of FROMKEY – TOKEY – COUNT.

Here is an example of repro for one key.

```
//CKDS    DD DISP=SHR,DSN=SYS1.SC75NEW.SCSFCKDS
//SYSPRINT DD SYSOUT=*
//OUTPUT  DD SYSOUT=*
//SEQ DD DSN=PE06.REPRO5.OUT,UNIT=SYSDA,
// SPACE=(TRK,(10,10)),
// DISP=(NEW,CATLG),
// DCB=(LRECL=372,BLKSIZE=2048,RECFM=VB)
//SYSIN   DD *,LRECL=80
 REPRO INFILE(CKDS) OUTFILE(SEQ) -
 FROMKEY (DATASET.PE06.ICSF.ENCRYPT.ME.ENCRKEY.00000005) -
 COUNT (1)
/*
//
 REPRO INFILE(CKDS) OUTFILE(OUTPUT) -
 FROMKEY (DATASET.PE06.ICSF.ENCRYPT.ME.ENCRKEY.00000001) -
 TOKEY   (DATASET.PE06.ICSF.ENCRYPT.ME.ENCRKEY.00000010) -
COUNT(10)
/*
//
```

Browse to verify the content of the REPRO SEQ file

```
BROWSE    PE06.REPRO5.OUT                        Line 0000000000 Col 001 080
 Command ===>                                                Scroll ===> CSR
******************************** Top of Data ********************************
DATASET.PE06.ICSF.ENCRYPT.ME.ENCRKEY.00000005                 DATA    ........
******************************* Bottom of Data ******************************
```

__ b. Send the dataset to your remote site B

__ c. On site B repro back in the key from the SEQ file into the CKDS

```
//JS010  EXEC PGM=IDCAMS
//CKDS    DD DISP=SHR,DSN=SYS1.SC60NEW.SCSFCKDS
//SYSPRINT DD SYSOUT=*
//OUTPUT  DD SYSOUT=*
//SEQ DD DSN=PE06.REPRO5.OUT,DISP=SHR UNIT=SYSDA,
//*SPACE=(TRK,(10,10)),
//*DISP=(NEW,CATLG),
//*DCB=(LRECL=372,BLKSIZE=2048,RECFM=VB)
//SYSIN   DD *,LRECL=80
 REPRO INFILE(SEQ) OUTFILE(CKDS)
/*
//
```

__ d. then refresh

```
//EUTIL  EXEC PGM=CSFEUTIL,
//       PARM='SYS1.SC60NEW.SCSFCKDS,REFRESH'
//CSFDIAG DD SYSOUT=*,LRECL=133
//CSFKEYS DD SYSOUT=*,LRECL=1044
//CSFSTMNT DD SYSOUT=*,LRECL=80
```

__ e. Check that ICSF is happy and can list this new key label ;-

```
Active CKDS: SYS1.SC60NEW.SCSFCKDS                          Keys: 14

 Action characters: A, D, K, M, P, R  See the help panel for details.
 Status characters: - Active   A Archived   I Inactive

 Select the records to be processed and press ENTER
 When the list is incomplete and you want to see more labels, press ENTER
 Press END to return to the previous menu

 A S Label     Displaying 1       to 14     of 14                  Key Type
 ------------------------------------------------------------------------------
 _ - AAAA.FIRST.KEY                                                 DATA
 _ - DATASET.ENCRYPTKEY.001                                         DATA
 _ - DATASET.PE01.TEST                                              DATA
 _ - DATASET.PE01.TESTNEWGEN                                        DATA
 _ - DATASET.PE01.TESTNEWKEY                                        DATA
 _ A DATASET.PE03.AC01                                              DATA
 _ - DATASET.PE06.ICSF.ENCRYPT.ME.ENCRKEY.00000005                  DATA
 _ - KEY LABEL.ANDY.COU1                                            DATA
```

__ f. Define the profile in CSFKEYS for this new key label

```
/*-------------------------------------------------------------------*/
RDEFINE CSFKEYS DATASET.PE06.ICSF.ENCRYPT.ME.ENCRKEY.00000005        +
          UACC(NONE)
PERMIT DATASET.PE06.ICSF.ENCRYPT.ME.ENCRKEY.00000005                 +
  CLASS(CSFKEYS) ID(PE06)   +
```

```
  ACCESS(READ)
  /*           WHEN(CRITERIA(SMS(DSENCRYPTION))) */
/*PE DATASET.PE06.ICSF.ENCRYPT.ME.ENCRKEY.00000005   +               */
/*   CLASS(CSFKEYS) ID(PE06) DEL*/

/*-------------------------------------------------------------------*/
/* The resource must specify the ICSF segment keywords to be able to */
/* use the key label for protected key.                              */
/*-------------------------------------------------------------------*/
RALTER CSFKEYS DATASET.PE06.ICSF.ENCRYPT.ME.ENCRKEY.00000005          +
  ICSF(SYMCPACFWRAP(YES) SYMCPACFRET(YES))

/*-------------------------------------------------------------------*/
/* Verify that the ICSF segment contains the protected key fields.   */
/*-------------------------------------------------------------------*/
RLIST CSFKEYS DATASET.PE06.ICSF.ENCRYPT.ME.ENCRKEY.00000005         +
 ICSF NORACF

/*-------------------------------------------------------------------*/
/* Permit the data owner to use the key when accessed through DFSMS  */
/*-------------------------------------------------------------------*/
PERMIT DATASET.PE06.ICSF.ENCRYPT.ME.ENCRKEY.00000005                  +
 CLASS(CSFKEYS) ID(PRICHAR)    +
  ACCESS(READ) WHEN(CRITERIA(SMS(DSENCRYPTION)))

/*-------------------------------------------------------------------*/
/* Refresh the CSFKEYS class to ensure that all users, started       */
/* tasks, and jobs are referencing the updated resource.             */
/*-------------------------------------------------------------------*/
SETROPTS RACLIST(CSFKEYS)
SETROPTS RACLIST(CSFKEYS) REFRESH

/*-------------------------------------------------------------------*/
/* Verify the data owner user id is in the conditional access list   */
/*-------------------------------------------------------------------*/
RLIST CSFKEYS DATASET.PE06.ICSF.ENCRYPT.ME.ENCRKEY.00000005       +
     AUTHUSER
```

__ g. Export encrypted dataset from site A to site B (dump site A /restore in site B)

__ h. Try to access encrypted dataset in site B

```
BROWSE    PE06.ICSF.ENCRYPT.ME.DATA5
 Command ===>
******************************** Top of Data *
Hello ITSO world
Have fun with Pervasive encryption
******************************** Bottom of Data
```

So that scenario works very well if you never change the content of the SEQ file containing the key with the label, and you import the initial record with its existing key label.

We tried to use that same method and and update the key label in the sequential file, in the case where the same key label already exists at the partner's site, but it turns out tha any time you edit/update the SEQ file, when you repro back, the repro works fine, but ICSF sees the new entry as an invalid key and sends some error messages when you do the CKDS refresh:

Here what you get if i try to edit the sequential file wihich contains the repro'ed key, and if you try to edit the name of the label to change it.

```
CSFM533I CKDS RECORD 8 UNUSABLE AND SKIPPED, LABEL DATASET.PE06.ICSF.E
NCRYPT.ME.ENCRKEY.00000001.
```

There is some sanity check which prevents to import a record that contains a key that you update in the SEQuential file. Therefore the method of repro is ONLY usable if:

- the MASTER keys are the same
- the key label you want to exchange currently does not exist at the partner's site CKDS.

Option 2:

you can use a sample REXX program that invoked ICSF callable services (CSNBKRC (create key label) , CSNBKRR (CKDS read) and CSNBKRW (overwrite key label) for CKDS transfers).

You can obtain and download a sample tool at:

ftp://ftp.software.ibm.com/s390/zos/tools/keyxfer/-

The key transfer tool (KEYXFER) is a REXX exec that runs on MVS.

KEYXFER facilitates the transfer of PKDS or CKDS key tokens between systems that use the Integrated Cryptographic Services Facility (ICSF).

The KEYXFER tool assumes the following:

__ 1. 1. ICSF is running on the systems involved in the key transfer

__ 2. 2. ICSF has an active Key Data Set (CKDS/PKDS)

For a symmetric key token (as used by AES dataset encryption) transfer the tool retrieves the token from the active CKDS and writes it to a data set (file).

The data set 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 file and store it into the active PKDS or CKDS.

The tokens are referenced by label.

Transferring the key token requires that the receiving systems use the same ICSF master key.

The key in CKDS is protected by the master key. It can't leave the CKDS in clear form. When it is transferred to another CKDS, it is still encrypted by the master key of the source CKDS. In order for the target CKDS to access the key, it needs the same master key to decrypt.

If ICSF services are RACF protected (CSFSERV) then access will be required by the user for the CSNBKRC (create key label) , CSNBKRR (CKDS read) and CSNBKRW (overwrite key label) for CKDS transfers.

Sample invocations:

KEYXFER WRITE_CKDS, CKDS.KEY.LABEL, TEMP.MEM

write the key token stored in the active CKDS under the label CKDS.KEY.LABEL to the data set TEMP.

KEYXFER READ_CKDS, CKDS.KEY.LABEL, TEMP.MEM

- read the key token contained in the data set TEMP.MEM and write the token to the active CKDS under the label CKDS.KEY.LABEL (If the label already exists in the CKDS the operation will fail.)

KEYXFER READ_CKDS, CKDS.KEY.LABEL, TEMP.MEM, OVERWRITE

- read the key token contained in the data set EMP.MEM and write the token to the active CKDS under the label CKDS.KEY.LABEL (If the label already exists in the CKDS the token for that label will Overwritten)

KEYXFER READ_CKDS, , TEMP.MEM

- read the key token contained in the data set TEMP.MEM and write the token to the active CKDS. Since no LABEL was specified the label Contained in the file is used as the token on the new system

Here all the details of our procedure:

- make sure you copy the keyxfer rexx utility in one of the datasets of your logon SYSPROC/SYSEXEC Concatenation.
- allocate a dataset PE06.ICSF.CSFKEYS

```
Data Set Name . . . . : PE06.ICSF.CSFKEYS

General Data                          Current Allocation
 Management class . . : **None**       Allocated cylinders : 5
 Storage class  . . . : **None**       Allocated extents . : 1
  Volume serial . . . : CONTS4
  Device type . . . . : 3390
 Data class . . . . . : **None**
  Organization  . . . : PS            Current Utilization
  Record format . . . : FB             Used cylinders  . . : 1
  Record length . . . : 512            Used extents  . . . : 1
  Block size  . . . . : 5120
  1st extent cylinders: 5
  Secondary cylinders : 5             Dates
```

- We are going to export (write) a key with its key label: create a member (ex: $KEYXFWR) in one of your JCL libraries containg:

```
BROWSE    PE06.JCL($KEYXFWR) - 01.02              Line 0000000000 Col
 Command ===>                                                Scroll =
******************************** Top of Data ************************
KEYXFER WRITE_CKDS, DATASET.PE06.ICSF.ENCRYPT.ME.ENCRKEY.00000005   ,+
PE06.ICSF.CSFKEYS
```

- This member indicates that we want to export key:
  - DATASET.PE06.ICSF.ENCRYPT.ME.ENCRKEY.00000005
- Invoke this member by entering EX(execute) in front of the member name:

```
BROWSE             PE06.JCL
Command ===>
            Name      Prompt        S
ex_______ $KEYXFWR *Browsed
```

- You will get the following messages on your TSO screen:

```
> 11/13/17  9:58am
 > DATASET.PE06.ICSF.ENCRYPT.ME.ENCRKEY.00000005   written to  'PE06.ICSF.CSFKE
YS'
***
```

- You can view the content of dataset 'PE06.ICSF.CSFKEYS

```
BROWSE    PE06.ICSF.CSFKEYS                        Line 000000000
Command ===>                                                 Scr
******************************** Top of Data ********************
 > 11/13/17  10:00am
DATASET.PE06.ICSF.ENCRYPT.ME.ENCRKEY.00000005
010000000400C01E49232659E5B39664D67CF764BE9E97A6E084588633B11F1C
CA9B482C96F07867AE13357F627E23D40000000000000000000100002050459D8D
```

Now is the time to send this file to your remote site to import the key.

On the remote site, depending if the key label is already defined or not, you will have to use 2 flavors of the invocation of keyxfer.rexx:

__ a. Copy the keyxfer rexx tool in a SYSPROC/SYSEXEC Concatenation

__ b. If the key label does not currently exist, then create a member with:

```
EDIT       PE06.JCL($KEYXFRD) - 01.01                 Columns 00001 0007
Command ===>                                            Scroll ===> DAT
****** ***************************** Top of Data ***************************
000001 KEYXFER READ_CKDS,
DATASET.PE06.ICSF.ENCRYPT.ME.ENCRKEY.00000005   ,+
000002 PE06.ICSF.CSFKEYS
```

This will read the key and key label from dataset PE06.ICSF.CSFKEYS, and will invoke ICSF to create the corresponding entry in the active CKDS:

__ c. Invoke the member by entering EX in front of the member name:

```
EDIT              PE06.JCL
Command ===>
           Name     Prompt       Size   Cre
ex_______ $KEYXFRD *Edited          2  2017
```

you will get the following messages on your TSO creen:

```
> 11/13/17  10:09am
 > DATASET.PE06.ICSF.ENCRYPT.ME.ENCRKEY.00000005  created
 > DATASET.PE06.ICSF.ENCRYPT.ME.ENCRKEY.00000005  overwritten
 > 'PE06.ICSF.CSFKEYS'  processed successfully.
***
```

If you use ICSF ISPF utility , you can now see that the key label has been created:

```
Active CKDS: SYS1.SC60NEW.SCSFCKDS                          Keys: 14

 Action characters: A, D, K, M, P, R  See the help panel for details.
 Status characters: - Active   A Archived   I Inactive

 Select the records to be processed and press ENTER
 When the list is incomplete and you want to see more labels, press ENTER
 Press END to return to the previous menu

 A S Label     Displaying 1       to 14     of 14                  Key Type
 ------------------------------------------------------------------------------
 _ - AAAA.FIRST.KEY                                                 DATA
 _ - DATASET.ENCRYPTKEY.001                                         DATA
 _ - DATASET.PE01.TEST                                              DATA
 _ - DATASET.PE01.TESTNEWGEN                                        DATA
 _ - DATASET.PE01.TESTNEWKEY                                        DATA
 _ I DATASET.PE03.AC01                                              DATA
 _ - DATASET.PE06.ICSF.ENCRYPT.ME.ENCRKEY.00000005                  DATA
```

__ d. Next send the encrypted dataset from site A to site B, and try to browse/edit it.

__ e. It is successful !

```
BROWSE    PE06.ICSF.ENCRYPT.ME.DATA5
 Command ===>
******************************** Top of Data *
Hello ITSO world
Have fun with Pervasive encryption
```

```
******************************** Bottom of Data
```

If you try to invoke the keyxfer tool and import a key label that already exists, then the tool will fail the operation withy message:

```
> 11/13/17  10:29am
*ERROR*  DATASET.PE06.ICSF.ENCRYPT.ME.ENCRKEY.00000005  exists
         - overwrite option has not been specified
***
```

You will have to specify the overwrite option to force the existing key label to be replaced (overwritten)

```
KEYXFER READ_CKDS, DATASET.PE06.ICSF.ENCRYPT.ME.ENCRKEY.00000005   ,+
PE06.ICSF.CSFKEYS , OVERWRITE
```

And then the result would be :

```
> 11/13/17  10:32am
 > DATASET.PE06.ICSF.ENCRYPT.ME.ENCRKEY.00000005  overwritten
 > 'PE06.ICSF.CSFKEYS'  processed successfully.
***
```

Scenario 2

Site A and B have different master keys, and want to exchange datasets encrypted with same key labels.

Site A and B have different master keys, and currently use different key labels, i.e the key KeyA that SITEA wants to send to SITEB is not currently used or defined in SITEB .

Below is a chart that depicts that configuration in our context:

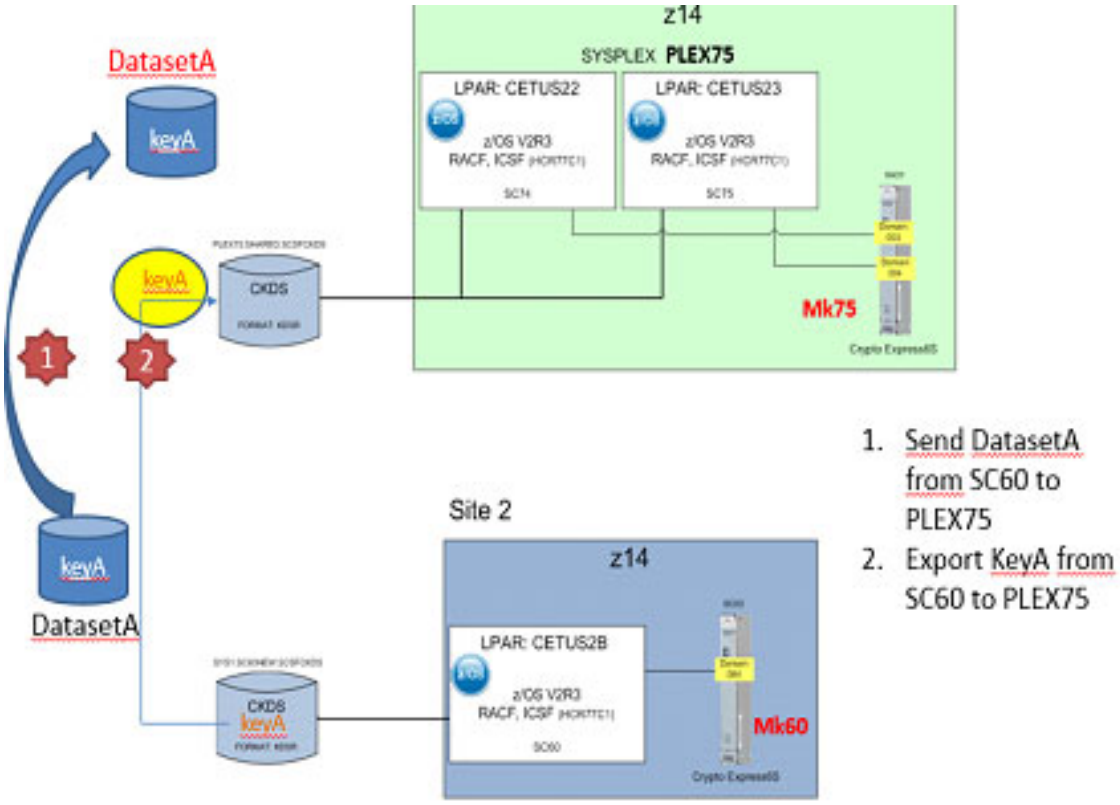

Figure 2-2. Transporting AES DATA keys between LPARs or sites that have different MKs

IBM Z uses the concept of a cryptographic domains to virtualize the physical coprocessor and enable sharing across multiple System z logical partitions (LPARs). A coprocessor can be shared by multiple LPARs and different types of operating systems.

Secure Keys generated and wrapped with master key of one domain are not usable by another domain using different master key. In the case where there is a need to transmit

encrypted data set to, perhaps, a business partner, that business partner will also need to have access to the DATA key used to encrypt the data sets.

To be able to access encrypted data on a system that has a different master key to the system on which the encrypted data sets are created, the secure AES DATA key needs to be transported from the creating, sending, system to this second, receiving, system. Currently, there is no built-in tool or process in z/OS or ICSF to allow an easy transport of AES DATA keys. However, there is a set of sample Rexx codes that are available at following URL that may be used to perform this AES DATA key transportation between LPARs at your site, or between your organization and business partners, that have different master keys:

https://www.ibm.com/developerworks/community/blogs/79c1eec4-00c4-48ef-ae2b-01bd8448dd6c/entry/Rexx_Samples_Transporting_an_AES_DATA_Key_using_an_ECC_Key_Pair?lang=en

The 6 Rexx samples that you need to download, and their functions are:

GENECC2.rexx	- Generate ECC private/public key pair
IMPRTEC2.rexx	- Store partner's ECC public key in PKDS
DRVAESXP.rexx	- Derive AES EXPORTER key on sending system from private key and receiving system's public key
DRVAESMP.rexx	- Derive AES IMPORTER key on receiving system from private key and sending system's public key
EXPAES32.rexx	- Translate the AES DATA key to an AES Cipher key then export the AES Cipher key under the AES EXPORTER key
IMPAES32.rexx	- Import the AES Cipher key using the AES IMPORTER key, and translate the AES Cipher key back to an AES DATA key

For our exercise, we will transport an AES DATA key from the monoplex system SC60, our sending system, to sysplex PLEX75 (SC74/SC75), our receiving system.

On the sending system, SC60, the Rexx samples that need to be executed are GENECC2, IMPRTEC2, DRVAESXP & EXPAES32, while on the receive system, PLEX75 (SC74 or SC75), the Rexx to be executed are GENECC2, IMPRTEC2, DRVAESMP & IMPAES32.

NOTE that minor alterations will need to be made to EXPAES32 and IMPAES32 prior to their execution. This will be shown later as part of the process being documented here.

After downloading the Rexx samples, we transferred the required Rexx, with the '.rexx' suffix removed, to SC60 & SC74/SC75 into 'PE08.REXX'.

What we did to demonstrate the transport of an AES DATA key from SC60 to PLEX75 (SC74/SC75):

- Create key label for encryption
- Define RACF profile to grant access to key label
- Create simple physical sequential (PS) data set encrypted with key label (AES DATA key) created
- DFSMSdss DUMP encrypted data set on SC60
- Transfer DSS dump data set from SC60 to PLEX75 (SC74/75)
- DFSSMSdss RESTORE encrypted data set on PLEX75 (SC74/SC75)
- Attempt to access encrypted data set (with result error due to missing key label)
- Execute REXX programs to transport the AES DATA key from SC60 to PLEX75 (SC74/SC75)
- Access encrypted data set on PLEX75 (SC74/SC75)

__ 1. Create key label for encryption

```
//STEP1  EXEC PGM=CSFKGUP
//CSFCKDS DD DISP=OLD,DSN=SYS1.SC60NEW.SCSFCKDS
//CSFIN   DD *,LRECL=80 ADD TYPE(DATA)
ALGORITHM(AES),  LABEL(DATASET.ENCRYPTKEY.001) LENGTH(32)
//CSFDIAG DD SYSOUT=*,LRECL=133
//CSFKEYS DD SYSOUT=*,LRECL=1044
//CSFSTMNT DD SYSOUT=*,LRECL=80
//*
//IF0001  IF (STEP1.RC EQ 0) THEN
//STEP2  EXEC PGM=CSFEUTIL,
//       PARM='SYS1.SC60NEW.SCSFCKDS,REFRESH'
//END0001 ENDIF
```

Figure 2-3. JCL used to run the KGUP to create the AES DATA key and key label DATASET.ENCRYPTKEY.001 and refresh the in-storage CKDS using CSFEUTIL

```
15.32.47 JOB06275  CSFM653I CKDS LOADED 18 RECORDS WITH AVERAGE SIZE 265
  .
  .
KEY GENERATION DIAGNOSTIC REPORT
ADD TYPE(DATA) ALGORITHM(AES),
 LABEL(DATASET.ENCRYPTKEY.001) LENGTH(32)
>>>CSFG0321 STATEMENT SUCCESSFULLY PROCESSED.
>>>CSFG0780 A REFRESH OF THE IN-STORAGE CKDS IS NECESSARY TO ACTIVATE
CHANGES MADE BY KGUP.
```

```
>>>CSFG0002 CRYPTOGRAPHIC KEY GENERATION - END OF JOB.  RETURN CODE = 0.
```

Figure 2-4. Output from KGUP to create the AES DATA key and key label:

__ 2. Define RACF profile to grant access to key label

- Define profile in CSFKEYS for new key label with ICSF segment

```
RDEFINE CSFKEYS DATASET.ENCRYPTKEY.001 UACC(NONE)
ICSF(SYMCPACFWRAP(YES) SYMCPACFRET(YES))
```

- Permit the data owner to use the key when accessed through DFSMS

```
PERMIT DATASET.ENCRYPTKEY.001 CLASS(CSFKEYS) ID(PE08) ACCESS(READ)
WHEN(CRITERIA(SMS(DSENCRYPTION)))
```

- Refresh the CSFKEYS class

```
SETROPTS RACLIST(CSFKEYS) REFRESH
```

- Verify that the ICSF segment contains the protected key fields and data owner is in the conditional access list

```
RLIST CSFKEYS DATASET.ENCRYPTKEY.001 ICSF AUTHUSER
```

- Create a generic DATASET resource to protect our encrypted data set

```
ADDSD 'PE08.SC60.ENCRYPT.*' UACC(NONE)
```

- Specify the encryption key label in the DFP segment

```
ALTDSD 'PE08.SC60.ENCRYPT.*' DFP(DATAKEY(DATASET.ENCRYPTKEY.001))
```

- Verify that the key label is in the DFP segment

```
LISTDSD DATASET('PE08.SC60.ENCRYPT.*') DFP NORACF
```

__ 3. Create simple physical sequential (PS) data set encrypted with key label (AES DATA key) created

We used the following JCL to create a small encrypted PS data set using IEBDG:

```
//STEP1    EXEC PGM=IEBDG
//SYSPRINT DD  SYSOUT=*
//OUTDATA  DD DISP=(,CATLG),DSN=PE08.SC60.ENCRYPT.DATASET,
//         UNIT=SYSDA,SPACE=(TRK,(5,1),RLSE),
//         DCB=(LRECL=80,RECFM=FB,BLKSIZE=0,DSORG=PS),
//         DSNTYPE=EXTREQ
//SYSIN    DD  *
   DSD     OUTPUT=(OUTDATA)
   FD      NAME=HELLO,LENGTH=21,PICTURE=21,'HELLO WORLD FROM SC60'
   CREATE  QUANTITY=1,NAME=(HELLO),FILL=X'40'
   END
/*
```

Figure 2-5. JCL to create a small encrypted PS data set using IEBDG

We can see the data set under ISPF option 3.4:

```
DSLIST - Data Sets Matching PE08                                Row 10 of 19
Command ===>                                                  Scroll ===> CSR

Command - Enter "/" to select action                Message          Volume
------------------------------------------------------------------------------
         PE08.SC60.ENCRYPT.DATASET                                     CONSM1
         PE08.SC60.ISPF42.ISPPROF                                      CONTS1

***************************** End of Data Set list ***************************
```

Browse of the data shows the unencrypted data (as the ID in use, PE08, has access to the key label):

```
BROWSE    PE08.SC60.ENCRYPT.DATASET              Line 0000000000 Col 001 080
 Command ===>                                                Scroll ===> CSR
******************************** Top of Data ********************************
HELLO WORLD FROM SC60
******************************** Bottom of Data *****************************
```

Data set Information shows that it's an EXTENDED format data set:

```
                              Data Set Information
Command ===>

Data Set Name . . . . : PE08.SC60.ENCRYPT.DATASET

General Data                          Current Allocation
 Management class . . : **None**       Allocated tracks  . : 1
 Storage class  . . . : DEFAULT        Allocated extents . : 1
  Volume serial . . . : CONSM1
  Device type . . . . : 3390
 Data class . . . . . : EFEA
  Organization  . . . : PS            Current Utilization
  Record format . . . : FB             Used tracks . . . . : 1
  Record length . . . : 80             Used extents  . . . : 1
  Block size  . . . . : 27920
  1st extent tracks . : 1
  Secondary tracks  . : 1             Dates
  Data set name type  : EXTENDED       Creation date . . . : 2017/11/13
                                       Referenced date . . : 2017/11/13
                                       Expiration date . . : ***None***
  SMS Compressible  . : NO
```

Figure 2-6. Data set Information shows that it's an EXTENDED format data set

IDCAMS LISTCAT of the dataset shows it's encrypted, the key label and EXTENDED format:

```
NONVSAM ------- PE08.SC60.ENCRYPT.DATASET
     IN-CAT --- UCAT.CONCAT
     HISTORY
        DATASET-OWNER-----(NULL)      CREATION--------2017.317
        RELEASE----------------2      EXPIRATION------0000.000
        ACCOUNT-INFO-----------------------------------(NULL)
      SMSDATA
        STORAGECLASS ----DEFAULT      MANAGEMENTCLASS---(NULL)
        DATACLASS ----------EFEA      LBACKUP ---0000.000.0000
      ENCRYPTIONDATA
        DATA SET ENCRYPTION----(YES)
        DATA SET KEY LABEL-----DATASET.ENCRYPTKEY.001
      VOLUMES
        VOLSER------------CONSM1      DEVTYPE------X'3010200F'      FSEQN---------
---------0
      ASSOCIATIONS--------(NULL)
      ATTRIBUTES
        VERSION-NUMBER---------2
        STRIPE-COUNT-----------1
        EXTENDED
NONVSAM ------- PE08.SC60.ENCRYPT.DATASET
     IN-CAT --- UCAT.CONCAT
     HISTORY
        DATASET-OWNER-----(NULL)      CREATION--------2017.317
        RELEASE----------------2      EXPIRATION------0000.000
        ACCOUNT-INFO-----------------------------------(NULL)
      SMSDATA
        STORAGECLASS ----DEFAULT      MANAGEMENTCLASS---(NULL)
        DATACLASS ----------EFEA      LBACKUP ---0000.000.0000
      ENCRYPTIONDATA
        DATA SET ENCRYPTION----(YES)
        DATA SET KEY LABEL-----DATASET.ENCRYPTKEY.001
      VOLUMES
        VOLSER------------CONSM1      DEVTYPE------X'3010200F'      FSEQN---------
---------0
      ASSOCIATIONS--------(NULL)
      ATTRIBUTES
        VERSION-NUMBER---------2
        STRIPE-COUNT-----------1
        EXTENDED
```

Figure 2-7. IDCAMS LISTCAT of the dataset shows it's encrypted, the key label and EXTENDED format

__ 4. DFSMSdss DUMP encrypted data set on SC60

We used the following JCL to dump the encrypted data set, PE08.SC60.ENCRYPT.DATASET, using DFSMSdss:

```
//STEP1    EXEC PGM=ADRDSSU,REGION=0M
//SYSPRINT DD SYSOUT=*
```

```
//TAPE     DD DISP=(,CATLG),DSN=PE08.DSS.DUMP,
//         UNIT=SYSDA,SPACE=(TRK,(5,1),RLSE)
//SYSIN    DD *
 DUMP DATASET(INCLUDE(PE08.SC60.ENCRYPT.DATASET)) -
      OUTDD(TAPE)                                 -
      ALLD(*)                                     -
      ALLEXCP                                     -
      OPT(4)
/*


PAGE 0001     5695-DF175  DFSMSDSS V2R03.0 DATA SET SERVICES     2017.317 17:57
 DUMP DATASET(INCLUDE(PE08.SC60.ENCRYPT.DATASET)) -
      OUTDD(TAPE)                                 -
      ALLD(*)                                     -
      ALLEXCP                                     -
      OPT(4)
ADR101I (R/I)-RI01 (01), TASKID 001 HAS BEEN ASSIGNED TO COMMAND 'DUMP '
ADR109I (R/I)-RI01 (01), 2017.317 17:57:30 INITIAL SCAN OF USER CONTROL STATEMENTS
COMPLETED
ADR016I (001)-PRIME(01), RACF LOGGING OPTION IN EFFECT FOR THIS TASK
ADR006I (001)-STEND(01), 2017.317 17:57:30 EXECUTION BEGINS
ADR903I (001)-PSEDM(01), DUMP OF EXTENDED SEQUENTIAL DATA SET
PE08.SC60.ENCRYPT.DATASET WAS SUCCESSFUL. SIZE OF DATA SET DUMPED WAS
                         0000000000000050
ADR801I (001)-DTDSC(01), 2017.317 17:57:30 DATA SET FILTERING IS COMPLETE. 1 OF 1
DATA SETS WERE SELECTED: 0 FAILED SERIALIZATION
                         AND 0 FAILED FOR OTHER REASONS
ADR454I (001)-DTDSC(01), THE FOLLOWING DATA SETS WERE SUCCESSFULLY PROCESSED
                          PE08.SC60.ENCRYPT.DATASET
ADR006I (001)-STEND(02), 2017.317 17:57:30 EXECUTION ENDS
ADR013I (001)-CLTSK(01), 2017.317 17:57:30 TASK COMPLETED WITH RETURN CODE 0000
ADR012I (SCH)-DSSU (01), 2017.317 17:57:30 DFSMSDSS PROCESSING COMPLETE. HIGHEST
RETURN CODE IS 0000
```

Figure 2-8. Output from the DSS DUMP:

__ 5. Transfer DSS dump data set from SC60 to PLEX75 (SC74/75)

We used Connect:Direct to transfer the DSS DUMP data set , PE08.DSS.DUMP, from SC60 to PLEX75 (SC74/75). Other means for the transfer can be:

- TSO Transmit
- FTP, in binary

__ 6. DFSSMSdss RESTORE encrypted data set on PLEX75 (SC74/SC75)

We use the following JCL to restore the encrypted data set, PE08.SC60.ENCRYPT.DATASET, using DFSMSdss, and renamed it as PE08.PLEX75.ENCRYPT.DATASET:

```
//STEP1    EXEC PGM=ADRDSSU,REGION=0M
//SYSPRINT DD SYSOUT=*
//TAPE     DD DISP=OLD,DSN=PE08.DSS.DUMP
//SYSPRINT DD SYSOUT=*
//SYSIN    DD *
 RESTORE DATASET(INCLUDE(**)) -
  RENAMEU(PE08.SC60.ENCRYPT.DATASET,PE08.PLEX75.ENCRYPT.DATASET) -
         INDDNAME(TAPE)        -
         OUTDY(BH5DB2)         -
         STORCLAS(DB1L)        -
         CATALOG
/*
PAGE 0001     5695-DF175  DFSMSDSS V2R03.0 DATA SET SERVICES     2017.318
10:57
 RESTORE DATASET(INCLUDE(**)) -
  RENAMEU(PE08.SC60.ENCRYPT.DATASET,PE08.PLEX75.ENCRYPT.DATASET) -
         INDDNAME(TAPE)        -
         OUTDY(BH5DB2)         -
         STORCLAS(DB1L)        -
         CATALOG
ADR101I (R/I)-RI01 (01), TASKID 001 HAS BEEN ASSIGNED TO COMMAND 'RESTORE '
ADR109I (R/I)-RI01 (01), 2017.318 10:57:53 INITIAL SCAN OF USER CONTROL STATEMENTS
COMPLETED
ADR016I (001)-PRIME(01), RACF LOGGING OPTION IN EFFECT FOR THIS TASK
ADR006I (001)-STEND(01), 2017.318 10:57:53 EXECUTION BEGINS
ADR780I (001)-TDDS (01), THE INPUT DUMP DATA SET BEING PROCESSED IS IN LOGICAL DATA
SET FORMAT AND WAS CREATED BY Z/OS DFSMSDSS
                        VERSION 2 RELEASE 3 MODIFICATION LEVEL 0 ON 2017.317
17:57:30
ADR711I (001)-NEWDS(01), DATA SET PE08.SC60.ENCRYPT.DATASET HAS BEEN ALLOCATED WITH
NEWNAME PE08.PLEX75.ENCRYPT.DATASET USING
                        STORCLAS DB1L, DATACLAS EFEA, AND NO MGMTCLAS
ADR474I (001)-TDNVS(01), DATA SET PE08.PLEX75.ENCRYPT.DATASET CONSISTS OF 00000001
TARGET TRACKS AND 00000001 SOURCE TRACKS
ADR906I (001)-PSERM(01), RESTORE OF EXTENDED SEQUENTIAL DATA SET
PE08.PLEX75.ENCRYPT.DATASET WAS SUCCESSFUL. SIZE OF DATA SET
                        RESTORED WAS 0000000000000050
ADR489I (001)-TDLOG(01), DATA SET PE08.PLEX75.ENCRYPT.DATASET WAS RESTORED
ADR454I (001)-TDLOG(01), THE FOLLOWING DATA SETS WERE SUCCESSFULLY PROCESSED
                         PE08.SC60.ENCRYPT.DATASET
ADR006I (001)-STEND(02), 2017.318 10:57:53 EXECUTION ENDS
ADR013I (001)-CLTSK(01), 2017.318 10:57:53 TASK COMPLETED WITH RETURN CODE 0000
ADR012I (SCH)-DSSU (01), 2017.318 10:57:53 DFSMSDSS PROCESSING COMPLETE. HIGHEST
RETURN CODE IS 0000
```

__ 7. Attempt to access encrypted data set

Now that the encrypted data set from SC60 has been restored on PLEX75 (SC74/SC75), we can see it under ISPF option 3.4:

```
DSLIST - Data Sets Matching PE08                          Row 14 of 16
```

```
Command ===>                                                Scroll ===> CSR

Command - Enter "/" to select action                Message          Volume
------------------------------------------------------------------------------
        PE08.PLEX75.ENCRYPT.DATASET                                  BH5DB2
        PE08.SC74.ISPF42.ISPPROF                                     BH5ST1
        PE08.SC75.ISPF42.ISPPROF                                     BH5ST4
***************************** End of Data Set list ****************************
```

Data set Information shows basically the same details as on SC60:

Data Set Information

```
Command ===>

Data Set Name . . . . : PE08.PLEX75.ENCRYPT.DATASET

General Data                            Current Allocation
 Management class . . : **None**         Allocated tracks  . : 1
 Storage class  . . . : DB1L             Allocated extents . : 1
  Volume serial . . . : BH5DB2
  Device type . . . . : 3390
 Data class . . . . . : EFEA
  Organization  . . . : PS              Current Utilization
  Record format . . . : FB               Used tracks . . . . : 1
  Record length . . . : 80               Used extents  . . . : 1
  Block size  . . . . : 27920
  1st extent tracks . : 1
  Secondary tracks  . : 1               Dates
  Data set name type  : EXTENDED         Creation date . . . : 2017/11/13
                                         Referenced date . . : 2017/11/13
                                         Expiration date . . : ***None***
  SMS Compressible  . : NO
```

IDCAMS LISTCAT of the dataset shows the same encryption information as on SC60:

```
NONVSAM ------- PE08.PLEX75.ENCRYPT.DATASET
     IN-CAT --- UCAT.BH5CAT
     HISTORY
       DATASET-OWNER-----(NULL)      CREATION--------2017.317
       RELEASE----------------2      EXPIRATION------0000.000
       ACCOUNT-INFO-----------------------------------(NULL)
     SMSDATA
       STORAGECLASS -------DB1L      MANAGEMENTCLASS---(NULL)
       DATACLASS ----------EFEA      LBACKUP ---0000.000.0000
     ENCRYPTIONDATA
       DATA SET ENCRYPTION----(YES)
       DATA SET KEY LABEL-----DATASET.ENCRYPTKEY.001
     VOLUMES
       VOLSER------------BH5DB2      DEVTYPE------X'3010200F'      FSEQN----
-----
--------0
     ASSOCIATIONS--------(NULL)
```

```
ATTRIBUTES
  VERSION-NUMBER---------2
  STRIPE-COUNT-----------1
  EXTENDED
```

Since there is no RACF profile defined on PLEX75 for the key label DATASET.ENCRYPTKEY.001, initial attempt to browse the data set resulted in:

```
IEC150I 913-84,IGG0193V,PE08,IKJACCT,ISP14257,9B16,BH5DB2,
PE08.PLEX75.ENCRYPT.DATASET,
RC=X'00000008',RSN=X'00000000'
ICH408I USER(PE08    ) GROUP(SYS1    ) NAME(PERVASIVE ENCRYPTION)
  DATASET.ENCRYPTKEY.001 CL(CSFKEYS )
  INSUFFICIENT ACCESS AUTHORITY
  FROM * (G)
  ACCESS INTENT(READ   )  ACCESS ALLOWED(NONE   )
```

As on SC60, we defined the RACF profile for key label DATASET.ENCRYPTKEY.001

- Define profile in CSFKEYS for new key label with ICSF segment

```
RDEFINE CSFKEYS DATASET.ENCRYPTKEY.001 UACC(NONE)
ICSF(SYMCPACFWRAP(YES) SYMCPACFRET(YES))
```

- Permit the data owner to use the key when accessed through DFSMS

```
PERMIT DATASET.ENCRYPTKEY.001 CLASS(CSFKEYS) ID(PE08) ACCESS(READ)
WHEN(CRITERIA(SMS(DSENCRYPTION)))
```

- Refresh the CSFKEYS class

```
SETROPTS RACLIST(CSFKEYS) REFRESH
```

- Verify that the ICSF segment contains the protected key fields and data owner is in the conditional access list

```
RLIST CSFKEYS DATASET.ENCRYPTKEY.001 ICSF AUTHUSER
```

A second attempt to browse the data set, with CSFKEYS profile in place for key label DATASET.ENCRYPTKEY.001 resulted in:

```
IEC143I 213-85,IGG0193V,PE08,IKJACCT,ISP14267,9B16,BH5DB2,
PE08.PLEX75.ENCRYPT.DATASET,
RC=X'00000008',RSN=X'0000271C'
```

RC8, RSN271C indicated the key label not found, which is expected as we have yet to transport the key label, and associated DATA key from SC60.

__ 8. Execute REXX programs to transport the AES DATA key from SC60 to PLEX75 (SC74/SC75)

To transport an AES DATA key from the sending system (SC60) to the receiving system (PLEX75), we ran the sample Rexx, with some minor modifications, on each of the systems. 2 of the Rexx programs needed to be run on both systems, while other 4 only need to be run depending on whether it's the sending or receiving system. The table below shows which Rexx programs need to be executed on the sending system (SC60) and receiving system (PLEX75).

	GENECC2	IMPRTEC2	DRVAESXP	DRVAESMP	EXPAES32	IMPAES32
SC60	Yes	Yes	Yes	No	Yes	No
PLEX75 (SC74/SC75)	Yes	Yes	No	Yes	No	Yes

Note that the following 4 Rexx programs were used to generate the private/public key pairs, exchange public keys and derive the IMPORTER and EXPORTER keys using the private & public keys:

- GENECC2, IMPRTEC2, DRVAESXP, DRVAESMP

Once the public keys have been shared between sending and receiving systems, and the IMPORTER and EXPORTER keys have been created, you can export (Rexx EXPAES32) and import (Rexx IMPAES32) any DATA keys without the need to run these 4 Rexx programs again.

__ 9. Changes made to the Rexx samples

There are a number of variables related to labels in the Rexx samples:

GENECC2	IMPRTEC2	DRVAESXP	DRVAESMP	EXPAES32	IMPAES32
ecc_key_label	ecc_pubkey_label	sender_key_label receiver_key_label exporter_key_label	sender_key_label receiver_key_label importer_key_label	exporter_key_label aes_data_key_label	importer_key_label aes_data_key_label

'aes_data_key_label' is the key label of the AES data key that is being sent, so we changed it to key label that was used to create the encrypted dataset on SC60, DATASET.ENCRYPTKEY.001.

For the other label variables, they could have been left as-is. But for clarity, we updated them as follow to make them more identifiable as to which system they belong to:

ecc_key_label = SC60.ECC.PRIVATE.KEY, on SC60 =
PLEX75.ECC.PRIVATE.KEY, on PLEX75

ecc_pubkey_label = PLEX75.ECC.PUBLIC.KEY, on SC60

= SC60.ECC.PUBLIC.KEY, on PLEX75

sender_key_label = SC60.ECC.PRIVATE.KEY, on SC60

= SC60.ECC.PUBLIC.KEY, on PLEX75

receiver_key_label = PLEX75.ECC.PUBLIC.KEY, on SC60

= PLEX75.ECC.PRIVATE.KEY, on PLEX75

exporter_key_label = SC60.AES.EXPORTER.KEY, on SC60

importer_key_label = PLEX75.AES.IMPORTER.KEY, on PLEX75

A notable change is in Rexx EXPAES32 which is to be run on the sending system (SC60). As supplied, the Rexx will delete the AES DATA key you specify via 'aes_data_key_label', and generate an AES DATA key with that label and store it in the CKDS.

This is fine if you are creating a new AES DATA that has not been used to encrypt any data set, and sharing it with another system (or business partner). In our exercise, we are transporting an AES DATA that we have used to encrypt a data set.

Therefore, we do not want the key and label to be deleted, hence we commented out the 3 lines in the Rexx that "CALL CSNBKRD", "CALL CSNBKGN" and "CALL CSNBKRC2". Depending on your requirement, you can either leave these CALLs in place or comment them out as we did – be sure to comment them out if you are processing a key and label that is already in use.

Another potential change is with Rexx **IMPAES32** on the receiving system. The Rexx will, via "CALL CSNBKRD", delete the key label (specify via aes_data_key_label) that we are importing from the CKDS. For our exercise, we are sure the key and label that we are importing does not exist on the receiving system, so we left the CALL as-is. However, you MUST ensure on your receiving system that the key and label that you are importing does not already exist, and possibly in use – you do NOT want to delete an AES DATA key that is in use on your receiving system.

Note that if Granular Key Label Access Control is enabled, additional RACF CSFKEYS profiles for the key labels being created by the Rexx programs will be required. Refer to ICSF Administrator's Guide for details on Granular Key Label Access Control and how to enable it.

---

10. Required RACF CSFSERV profiles for Rexx samples

The table below details the RACF CSFSERV resource profiles that are required (with READ access) for successful execution of the Rexx samples.

Rexx Exec	Resource Name	Callable service description
GENECC2	CSFPKG CSFPKRC CSFPKRD CSFPKX	PKA Key Generate PKDS Record Create PKDS Record Delete PKA Public Key Extract
IMPRTEC2	CSFPKRC CSFPKRD CSFPKRR	PKDS Record Create PKDS Record Delete PKDS Record Read
DRVAESXP	CSFEDH CSFKRC2 CSFKRD CSFKRR2	ECC Diffie-Hellman Key Record Create2 Key Record Delete Key Record Read2
DRVAESMP	CSFEDH CSFKRC2 CSFKRD CSFKRR2	ECC Diffie-Hellman Key Record Create2 Key Record Delete Key Record Read2
EXPAES32	CSFKRR2 CSFKTR2 CSFKYT2 CSFSYX	Key Record Read2 Key Translate2 Key Test2 Symmetric Key Export
IMPAES32	CSFKRC2 CSFKRD CSFKTR2 CSFKYT2 CSFSYI2	Key Record Create2 Key Record Delete Key Translate2 Key Test2 Symmetric Key Import2

Figure 2-9. RACF CSFSERV resource profiles that are required (with READ access) for successful execution of the Rexx samples

__ 11. Executing the Rexx

We ran Rexx GENECC2 on both sending and receive systems to generate the private/public key pairs for each system. The private key will be stored into the ICSF PKDS. The public key will be output from the Rexx that will be used as input to IMPRTEC2.

__ 12. Generate private/public key pairs

On the sending system, SC60, execute the Rexx via EX 'PE08.EXEC(GENECC2)'.

Output from GENECC2 on SC60:

```
ecc public key length (hex) 0000009Bx
ecc public key
1E00009B00000000210000930000000000000209008504001608 7F6C5E91EC16
6FAAEC904652CEC3A58AE51ABC825FFCB4745D392F0899D9A874957487D4AF4A
087D69E13520AF20A7E50C669D2A617A450DD8FB86300C7F280147D0F57ED8E7
FE2DC33B93ED9BC2C7FF1154383D1518FDD2BFA37EB680339CD1900CF5519CF8
B297088D0516400676F9B14092DFFFC383DF79625790D534904867
------------------------------------------------------------------
End of Sample
------------------------------------------------------------------
```

On receiving sysplex, PLEX75, execute the Rexx via EX ‘PE08.EXEC(GENECC2)’.

```
ecc public key length (hex) 0000009Bx
ecc public key
1E00009B00000000210000930000000000000209008504002DB3764961C22132
7A540F282FE6CF0473BE51F835216426C6133D67C3307BF6704B764B120D9BB5
6648B6D1DB8BA4D81130C26C8F8DAE8709EEC200FFCC21538C014C821DADA329
D2C6A7879D89245119E1FC2E2BF5F6DCFD3F9BAE940F018830C191AFDD8C7302
97552C6BDBC6C3BE2628B455876D847B3A8DFF5F36A76B57BE5F7A
------------------------------------------------------------------
End of Sample
------------------------------------------------------------------
```

__ 13. Create ECC public key from partner’s public key

The next step is to run IMPRTEC2 to create an ECC public key from a partner's public key. We updated ecc_pubkey in IMPRTEC2 on SC60 with the output from execution of GENECC2 on PLEX75, thus:

```
ecc_pubkey = ,
'1E00009B00000000210000930000000000000209008504002DB3764961C22132'x||,
'7A540F282FE6CF0473BE51F835216426C6133D67C3307BF6704B764B120D9BB5'x||,
'6648B6D1DB8BA4D81130C26C8F8DAE8709EEC200FFCC21538C014C821DADA329'x||,
'D2C6A7879D89245119E1FC2E2BF5F6DCFD3F9BAE940F018830C191AFDD8C7302'x||,
'97552C6BDBC6C3BE2628B455876D847B3A8DFF5F36A76B57BE5F7A'x
```

Figure 2-10. Create ECC public key from partner’s public key

Similarly, ecc_pubkey in IMPRTEC2 on PLEX75 was updated with the output from execution of GENECC2 on SC60, thus:

```
ecc_pubkey = ,
'1E00009B00000000210000930000000000000209008504001608 7F6C5E91EC16'x||,
'6FAAEC904652CEC3A58AE51ABC825FFCB4745D392F0899D9A874957487D4AF4A'x||,
'087D69E13520AF20A7E50C669D2A617A450DD8FB86300C7F280147D0F57ED8E7'x||,
'FE2DC33B93ED9BC2C7FF1154383D1518FDD2BFA37EB680339CD1900CF5519CF8'x||,
'B297088D0516400676F9B14092DFFFC383DF79625790D534904867'x
```

On the sending system, SC60, we executed the Rexx via EX ‘PE08.EXEC(IMPRTEC2)’

Resulting output from IMPRETEC2 on SC60:

```
ecc key label  PLEX75.ECC.PUBLIC.KEY
ecc public key
1E00009B0000000021000093000000000000020900850400 2DB3764961C22132
7A540F282FE6CF0473BE51F835216426C6133D67C3307BF6704B764B120D9BB5
6648B6D1DB8BA4D81130C26C8F8DAE8709EEC200FFCC21538C014C821DADA329
D2C6A7879D89245119E1FC2E2BF5F6DCFD3F9BAE940F018830C191AFDD8C7302
97552C6BDBC6C3BE2628B455876D847B3A8DFF5F36A76B57BE5F7A
ecc public key length 155
ecc public key length (hex) 0000009Bx
------------------------------------------------------------------
End of Sample
------------------------------------------------------------------
```

Likewise, executing IMPRETEC2 on PLEX75:

```
ecc key label  SC60.ECC.PUBLIC.KEY
ecc public key
1E00009B0000000021000093000000000000020900850400 16087F6C5E91EC16
6FAAEC904652CEC3A58AE51ABC825FFCB4745D392F0899D9A874957487D4AF4A
087D69E13520AF20A7E50C669D2A617A450DD8FB86300C7F280147D0F57ED8E7
FE2DC33B93ED9BC2C7FF1154383D1518FDD2BFA37EB680339CD1900CF5519CF8
B297088D0516400676F9B14092DFFFC383DF79625790D534904867
ecc public key length 155
ecc public key length (hex) 0000009Bx
------------------------------------------------------------------
End of Sample
------------------------------------------------------------------
```

The ECC public keys are now stored in each system’s PKDS.

__ 14. Derive AES EXPORTER key on sending system

On sending system, SC60, we ran Rexx DRVAESXP to derive an AES EXPORTER key from the receiving system’s, PLEX75, ECC public key and its own private key.

Output from DRVAESXP:

```
derived AES EXPORTER key label: SC60.AES.EXPORTER.KEY

------------------------------------------------------------------
End of Sample
```

```
------------------------------------------------------------------
```

The EXPORTER key, SC60.AES.EXPORTER.KEY, is stored in the CKDS.

__ 15. Derive AES IMPORTER key on receiving system

On receiving sysplex, PLEX75, we ran Rexx DRVAESMP to derive an AES IMPORTER key from the sending system's, SC60, ECC public key and its own private key.

Output from DRVAESMP:

```
derived AES IMPORTER key label: PLEX75.AES.IMPORTER.KEY

------------------------------------------------------------------
End of Sample
------------------------------------------------------------------
```

The IMPORTER key, PLEX75.AES.IMPORTER.KEY, is stored in the CKDS.

__ 16. Export the AES DATA key on sending system

With the EXPORTER key derived, we are now ready to export the AES DATA key on SC60. The Rexx sample EXPAES32 translates the AES DATA key to an AES Cipher key then export the AES Cipher key under the AES EXPORTER key.

Note that the calls to CSNBKRD, CSNBKGN and CSNBKRC2 have been commented out in the Rexx sample since we did not want the Rexx to delete (and generate) the AES DATA key that is in used and being exported. The output from EXPAES32 forms the 'input' to IMPAES32 on the receiving system.

Output from EXPAES32:

```
verification pattern: 6E27120A7165770B
exported key:
020000880500000020213033A016F0896190000000000000000020201000100
001A000000000280000200010 2C000000003E0000000001409611F5998BBE716
3D625EE8C861E5611B73FE05F2250428B1E6304365DD8B2C5F588FF2AB0F231C
F46F0BF786F6A139F55EED7760EB883EDDE4FA608DC5C5510FCA36381B6ADA21
A0849C886F9DD316
exported key LENGTH:  136
exported key length (hex):  00000088x
------------------------------------------------------------------
End of Sample
------------------------------------------------------------------
```

__ 17. Import the AES DATA key on receiving system

To import the AES DATA, we ran Rexx sample IMPAES32 on PLEX75. IMPAES32 imports the AES Cipher key using the AES IMPORTER key, and translate the AES Cipher key back to to the original AES DATA key.

In additional the changes made to importer_key_label and aes_data_key_label mentioned earlier in this topic, verification_pattern and encrypted_key were updated with output from EXPAES32 on SC90, thus:

```
verification_pattern = '6E27120A7165770B'x
encrypted_key = ,
'020000880500000020213033A016F0896190000000000000000020201000100'x||,
'001A0000000028000020001020C000000003E0000000001409611F5998BBE716'x||,
'3D625EE8C861E5611B73FE05F2250428B1E6304365DD8B2C5F588FF2AB0F231C'x||,
'F46F0BF786F6A139F55EED7760EB883EDDE4FA608DC5C5510FCA36381B6ADA21'x||,
'A0849C886F9DD316'x
```

Output from IMPAES32:

```
Verification of the AES DATA KEY succeeded
-------------------------------------------------------------------
End of Sample
-------------------------------------------------------------------------------
```

We then went into the ICSF CKDS browser (ICSF Option 5.5;1) on PLEX75:

```
---------------------------- ICSF - CKDS KEYS List ----------- Row 1 to 5
of 5
COMMAND ===>                                                  SCROLL ===>
PAGE

Active CKDS: PLEX75.SHARED.SCSFCKDS                           Keys: 5

Action characters: A, D, K, M, P, R  See the help panel for details.
Status characters: - Active   A Archived   I Inactive

Select the records to be processed and press ENTER
When the list is incomplete and you want to see more labels, press ENTER
Press END to return to the previous menu

A S Label     Displaying 1       to 5     of 5                    Key
Type
------------------------------------------------------------------------------
----
_ - DATASET.ENCRYPTKEY.001                                        DATA
_ - ICSF.SECRET.AES256.KEY001                                     DATA
_ - PLEX75.AES.IMPORTER.KEY
IMPORTER
_ - SAMPLE.DERIVED.AES.IMPORTER.KEY
IMPORTER
```

```
       _ - SAMPLE.RECEIVED.AES.DATA.KEY                                DATA
```

Figure 2-11. ICSF CKDS browser (ICSF Option 5.5;1)

In addition to the AES DATA key, `DATASET.ENCRYPTKEY.001`, we have just imported, we can also see the IMPORTER key, `PLEX75.AES.IMPORTER.KEY`, that we created earlier with DRVAESMP.

__ 18. Access encrypted data set on PLEX75 (SC74/SC75)

With the CSFKEYS profiles for DATASET.ENCRYPTKEY.001 defined earlier, we browsed the data set on PLEX75 again, and success, we were able to see the unencrypted data as created on SC60:

```
BROWSE    PE08.PLEX75.ENCRYPT.DATASET             Line 0000000000 Col 001 080
 Command ===>                                                 Scroll ===> CSR
******************************** Top of Data ********************************
HELLO WORLD FROM SC60
******************************** Bottom of Data *****************************
```

We then ran a DSS PRINT of `PE08.PLEX75.ENCRYPT.DATASET` to show the data is still encrypted:

```
PAGE 0001     5695-DF175  DFSMSDSS V2R03.0 DATA SET SERVICES     2017.319
11:36
 PRINT DATASET(PE08.PLEX75.ENCRYPT.DATASET) INDYNAM(BH5DB2)
ADR101I (R/I)-RI01 (01), TASKID 001 HAS BEEN ASSIGNED TO COMMAND 'PRINT '
ADR109I (R/I)-RI01 (01), 2017.319 11:36:54 INITIAL SCAN OF USER CONTROL
STATEMENTS COMPLETED
ADR016I (001)-PRIME(01), RACF LOGGING OPTION IN EFFECT FOR THIS TASK
ADR006I (001)-STEND(01), 2017.319 11:36:54 EXECUTION BEGINS
*** TRACK(CCHH)  03A9000D         R0 DATA  0000000000000000
    COUNT  03A9000D01006D30
 0000  B6F1EA0D 69FBCE9E 9BEC95D1 2853B3BB ...
*.1........nJ....H...1...S³./><_-*
 0020  7D8CC3C7 F7768328 235F6E3A 483F1020 ...
*'.CG7.c..>........E...Z*³@$...*
 0040  BC0E0270 5728A162 D7A024D7 CAD31E4D ...
*......~.P..P.L.(.............*
 0060  00000000 00000000 00000000 00000000 ...
*................................*
 0080  TO  6CFF   SAME AS ABOVE
 6D00  00000000 00000000 00000000 00000000 ...
*.................&.............*
 6D20  00000000 00000000 0000E890 DA5A5AA5 ...  *.........Y..!!v
*
ADR006I (001)-STEND(02), 2017.319 11:36:54 EXECUTION ENDS
ADR013I (001)-CLTSK(01), 2017.319 11:36:54 TASK COMPLETED WITH RETURN CODE
0000
```

```
ADR012I (SCH)-DSSU (01), 2017.319 11:36:54 DFSMSDSS PROCESSING COMPLETE.
HIGHEST RETURN CODE IS 0000
```

Scenario 3:

Changing the key label scenario:

- In this scenarion both Site A and B have the same master key or different master key, and the key labl they want to exchange is already in use in both sites, i.e they have the key label KeyA already defined in both sites; in that case what would be the process to copy/export the encrypted key KeyA from site A to site B. Should i import the key under a different key label ?

We tried that scenario in a configuration with both sites having the same master key.
We updated the invocation of the keyxfer tool, so that the key should be imported under a different key label: DATASET.PE06.ICSF.ENCRYPT.ME.ENCRKEY.00000004; because key label 00000004 already existed in site B, we requested the overwrite option.

```
EDIT       PE06.JCL($KEYXFRD) - 01.03                    Columns 00001 000
Command ===>                                               Scroll ===> DA
****** ***************************** Top of Data ***************************
000001 KEYXFER READ_CKDS, DATASET.PE06.ICSF.ENCRYPT.ME.ENCRKEY.00000004   ,+
000002 PE06.ICSF.CSFKEYS , OVERWRITE
```

We then invoked the utility , and received messages:

```
> 11/13/17  10:54am
 > DATASET.PE06.ICSF.ENCRYPT.ME.ENCRKEY.00000004  created
 > DATASET.PE06.ICSF.ENCRYPT.ME.ENCRKEY.00000004  overwritten
 > 'PE06.ICSF.CSFKEYS'  processed successfully.
***
```

The CKDS ICSF browser utility nows shows that we have an additional key:

```
--------------------------- ICSF - CKDS KEYS List ----------- RECORDS DELETED
COMMAND ===>                                                 SCROLL ===> PAGE

Active CKDS: SYS1.SC60NEW.SCSFCKDS                           Keys: 17

Action characters: A, D, K, M, P, R  See the help panel for details.
Status characters: - Active   A Archived   I Inactive

Select the records to be processed and press ENTER
When the list is incomplete and you want to see more labels, press ENTER
Press END to return to the previous menu
```

```
A S Label     Displaying 1       to 17     of 17                    Key Type
--------------------------------------------------------------------------------
_ - AAAA.FIRST.KEY                                                   DATA
_ - DATASET.ENCRYPTKEY.001                                           DATA
_ - DATASET.PE01.TEST                                                DATA
_ I DATASET.PE01.TESTNEWGEN                                          DATA
_ - DATASET.PE01.TESTNEWKEY                                          DATA
_ I DATASET.PE03.AC01                                                DATA
_   <Record deleted>
_ - DATASET.PE06.ICSF.ENCRYPT.ME.ENCRKEY.00000004                    DATA
```

- Now what happens if I want to import on site B an encrypted dataset that was created on site A, and try to use it with this new key label DATASET.PE06.ICSF.ENCRYPT.ME.ENCRKEY.00000004 ?
- It was originally created on site A with key label DATASET.PE06.ICSF.ENCRYPT.ME.ENCRKEY.00000005<o:p\
- This key has been imported in site B under key label DATASET.PE06.ICSF.ENCRYPT.ME.ENCRKEY.00000004.
- I will dump/restore the dataset from site A to site B

Then listcat of the restored dataset shows:

```
NONVSAM ------- PE06.ICSF.ENCRYPT.ME.DATA5
     IN-CAT --- UCAT.CONCAT
     HISTORY
       DATASET-OWNER-----(NULL)      CREATION--------2017.316
       RELEASE----------------2      EXPIRATION------0000.000
       ACCOUNT-INFO-----------------------------------(NULL)
     SMSDATA
       STORAGECLASS ----DEFAULT      MANAGEMENTCLASS---(NULL)
       DATACLASS ----------EFEA      LBACKUP ---0000.000.0000
     ENCRYPTIONDATA
       DATA SET ENCRYPTION----(YES)
       DATA SET KEY LABEL-----DATASET.PE06.ICSF.ENCRYPT.ME.ENCRKEY.00000005
     VOLUMES
       VOLSER------------CONSM3      DEVTYPE------X'3010200F'      FSEQN---------
--------0
     ASSOCIATIONS--------(NULL)
     ATTRIBUTES
       VERSION-NUMBER---------2
       STRIPE-COUNT-----------1
       EXTENDED
***
```

- If I try to browse this dataset I get:

```
IEC143I 213-85,IGG0193V,PRICHAR,IKJACCNT,ISP07585,9642,CONSM3,
 PE06.ICSF.ENCRYPT.ME.DATA5,
 RC=X'00000008',RSN=X'0000271C'
```

***

- Because this key label DATASET.PE06.ICSF.ENCRYPT.ME.ENCRKEY.00000005 does not exist.
- If I try to alter the dataset name profile, like in the following example:

```
/*-------------------------------------------------------------------*/
/* Specify the encryption key label in the DFP segment.              */
/*-------------------------------------------------------------------*/
ALTDSD 'PE06.ICSF.ENCRYPT.ME.*'                                        +
  DFP(DATAKEY(DATASET.PE06.ICSF.ENCRYPT.ME.ENCRKEY.00000004))
```

- Then this has no effect on the existing dataset that you have exported/imported; in its catalog entry definition it still shows .

```
ENCRYPTIONDATA
   DATA SET ENCRYPTION----(YES)
   DATA SET KEY LABEL-----DATASET.PE06.ICSF.ENCRYPT.ME.ENCRKEY.00000005
```

- This new DFP segment definition will only affect datasets that will be newly created.

Note regarding key label name change:

If you import on site B a key label (keylab1) by changing its key label name (keylab2) , then be aware that any dataset which was created on site A with keylab1 will not be able to be exchanged and processed on site B if its corresponding key label name has been changed.
This would require on site A any form of rekeying of the dataset to key label keylab2, before being able to be sent over to site B, so that it can be used on site B with its new key label keylab2.

This was illustrated in the previous scenario.

Conclusion: You can export/import a key under a different key label, but this new key label can only be used to encrypt new datasets on site B, it can never be used to work with datasets which were created on site A and then transferred to site B.

This statement is true both for for exchanging keys between sites having the same master key, or different master keys.

With different key labels that would involve re-keying, which will complicates the process.

- With different key labels (say, KEYB), then you would have to re-key datasetA at SITE A from key label KEYA to KEYB, then send the re-keyed datasetA (now using key label KEYB) from SITE A to SITE B, and also transport key label KEYB from SITE A to SITE B .

References

Using new DFSMS functions in z/OS V2R3
https://www.ibm.com/support/knowledgecenter/SSLTBW_2.3.0/com.ibm.zos.v2r3.idak100/encryption23.htm

IBM Crypto Education Community
https://www.ibm.com/developerworks/community/groups/community/crypto

Dataset Encryption FAQs
https://www-03.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/FQ131494