IBM DB2 Native Encryption

Lab exercises

An IBM Proof of Technology

PoT.IM.15.1.000.01

Contents

THIS PAGE INTENTIONALLY LEFT BLANK

Lab preparation

This preparation has been done for you already if you are using an IBM-provided workstation with the VMware image on it.

If, however, you are using this guide on your own equipment and need to set up the labs yourself, you will need to do the following so you won't have to make any changes to the scripts:

Initial setup

__1. Use a Linux image of SuSE Linux Enterprise Server (SLES) 11.2 or higher

__2. Install IBM® DB2® 10.5 Fix Pack 5 or higher on your server

__3. Create a db2 instance called db2inst1

__4. Copy all the scripts into the home directory of the user db2inst1. These scripts will all be under this directory:

```
/home/db2inst1/db2_encrypt
```

It is best if you set the DB2 registry variable for communication:

```
db2set db2comm=tcpip
```

Also you will need to set your DBM configuration parameter with your appropriate service name:

```
db2 update dbm cfg using svcename 50000
```

Check the **`/etc/services`** file to make sure this service name matches what you set in the dbm cfg parm

Stop and start your instance to apply these changes.

__5. Review the Instructor's Guide to make sure your VM image setup is appropriate

** End of lab preparation for DB2 Native Encryption exercises

Lab 1 – IBM DB2 Native Encryption setup

1.1 - DB2 Native Encryption steps overview

This is a high level overview of the minimum steps required to implement native encryption in DB2:

DB2 Native Encryption setup steps	Command used:

Set the paths for the Global Security Kit	`export LD_LIBRARY_PATH=...`
	`export PATH=...`

Create a keystore *	`gsk8capicmd_64 -keydb -create -db [keystore]...`

Configure DB2 instance with the keystore information	`update dbm cfg using`
	`   keystore_type [keytype]`
	`   keystore_location [keypath]...`

Create the DB2 database using encryption	`create db [dbname] encrypt...`

*Note: We will only need to use a few of the **gsk8capicmd** utility commands for these lab exercises.

If you wish to see all the command options available for this utility, they can be found in the user guide named: **GSK_CapiCmd_UserGuide.pdf**

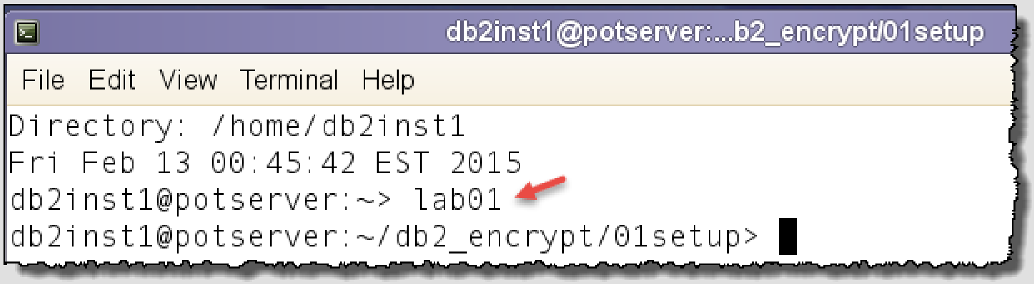
This guide is available with the other materials in this PoT asset.

1.2 - DB2 Global Security Kit (GSKit) path check

__1. Open the *db2inst1 terminal* (if not already open) and navigate to:

`/home/db2inst1/db2_encrypt/01setup`

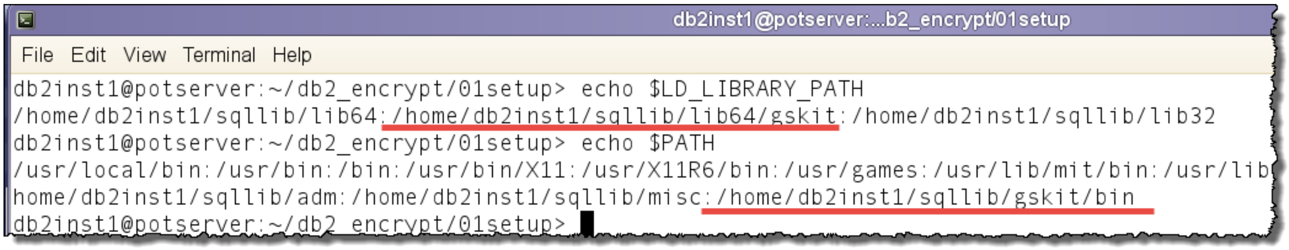
Note: To get to this lab directory easily, you can use alias: **`lab01`**


db2inst1@potserver:...b2_encrypt/01setup
File Edit View Terminal Help

```
Directory: /home/db2inst1
Fri Feb 13 00:45:42 EST 2015
db2inst1@potserver:~> lab01
db2inst1@potserver:~/db2_encrypt/01setup>
```

__2. Review the default path of environment variables by typing in the following:

```
echo $LD_LIBRARY_PATH
echo $PATH
```


db2inst1@potserver:...b2_encrypt/01setup
File Edit View Terminal Help

```
db2inst1@potserver:~/db2_encrypt/01setup> echo $LD_LIBRARY_PATH
/home/db2inst1/sqllib/lib64:/home/db2inst1/sqllib/lib64/gskit:/home/db2inst1/sqllib/lib32
db2inst1@potserver:~/db2_encrypt/01setup> echo $PATH
/usr/local/bin:/usr/bin:/bin:/usr/bin/X11:/usr/X11R6/bin:/usr/games:/usr/lib/mit/bin:/usr/lib
home/db2inst1/sqllib/adm:/home/db2inst1/sqllib/misc:/home/db2inst1/sqllib/gskit/bin
db2inst1@potserver:~/db2_encrypt/01setup>
```

Notice that the libraries for the 64-bit GSKit libraries have been created and the environment variable set for them during the installation of DB2. Shown highlighted are the paths that will be accessed during these lab exercises.

For UNIX, set these with export commands (if they are not already set for you during your DB2 install). You can put these export commands in either the instance owner's **`.profile`** or the file **`db2profile`**.

For Windows installation, include either of these in the **`PATH`** environment variable:

`C:\Program Files\IBM\gsk8\lib64` or

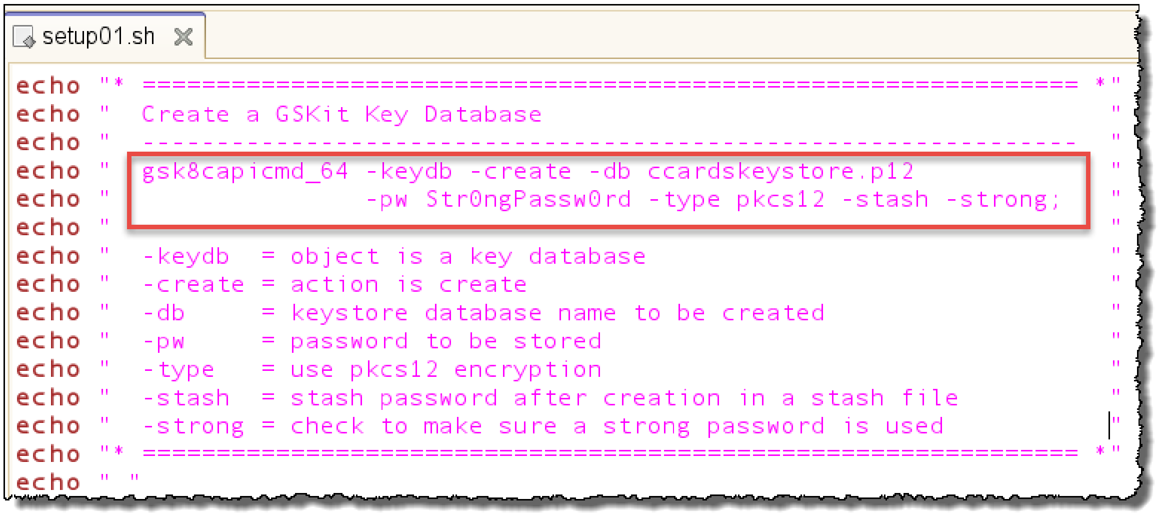
`C:\Program Files (x86)\IBM\gsk8\lib`

1.3 - Create a keystore (also known as a key database)

DB2 uses a storage object for PKCS#12-compliant encryption keys called a keystore (or key database). To create one for the first time, we will use the **`gsk8capicmd_64`** utility.

__3. Review script **`setup01.sh`**.

Notice the syntax for creating a keystore:

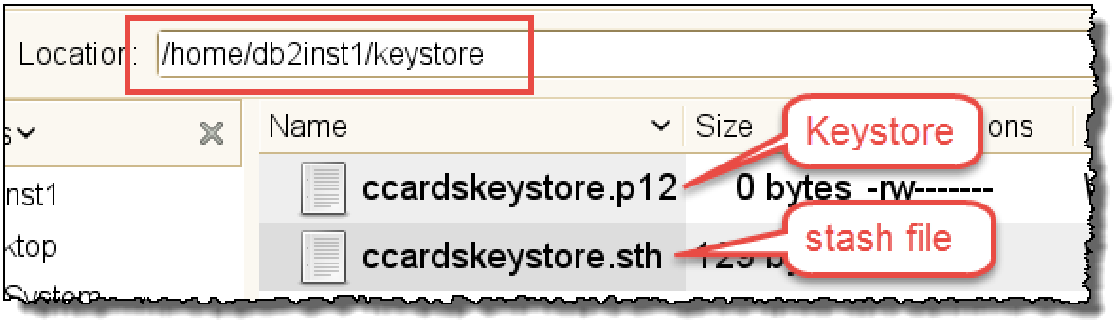


```
echo "* ============================================================== *"
echo "  Create a GSKit Key Database                                     "
echo "  --------------------------------------------------------------  "
echo "  gsk8capicmd_64 -keydb -create -db ccardskeystore.p12            "
echo "                 -pw Str0ngPassw0rd -type pkcs12 -stash -strong;  "
echo "                                                                  "
echo "  -keydb  = object is a key database                              "
echo "  -create = action is create                                      "
echo "  -db     = keystore database name to be created                  "
echo "  -pw     = password to be stored                                 "
echo "  -type   = use pkcs12 encryption                                 "
echo "  -stash  = stash password after creation in a stash file         "
echo "  -strong = check to make sure a strong password is used          "
echo "* ============================================================== *"
echo " "
```

__4. Run script **`setup01.sh.`**

`./setup01.sh`

Notice the two files created in the "keystore directory": the keystore itself and the stash file.



Location: /home/db2inst1/keystore

Name	Size	ons
ccardskeystore.p12	0 bytes	-rw-------
ccardskeystore.sth	129 b	

A stash file is used as an automatic way of providing a password. When accessing a key database, the system will first check for the existence of a stash file. If one exists, the contents of the file will be decrypted and used as input for the password.

The stash file can be read by only the file owner (in this case, the instance owner). If the password is not stashed, you cannot access an encrypted database until you provide the keystore password.

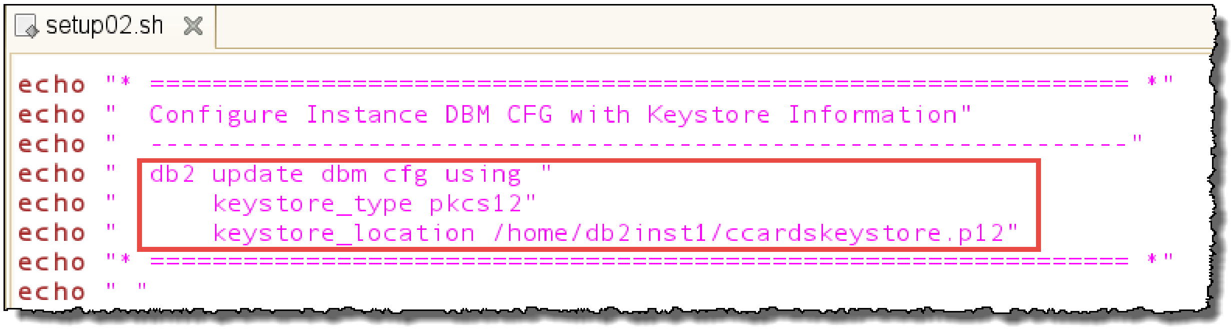
If you do not use a stash file, you must start the instance with an **`OPEN KEYSTORE USING`** clause. This allows many ways to provide a password, either with a prompt or a protected file.

1.4 - Configure the DB2 instance with the keystore information

Now we will point the DBM CFG parameters for our instance to know about the keystore:

__5. Review script **setup02.sh.**

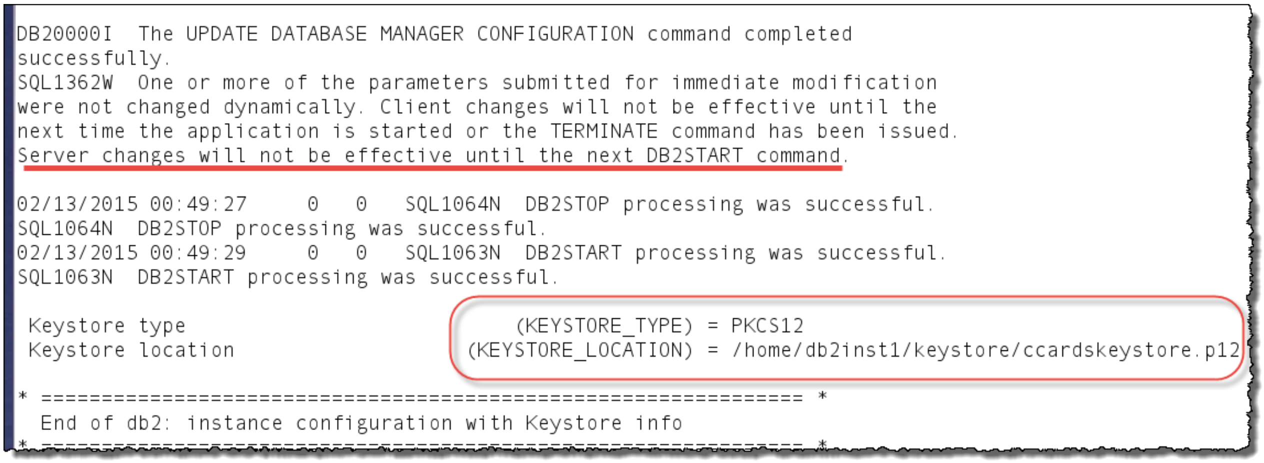
Notice that two DBM CFG parameters are used: **keystore_type** and **keystore_location**



```
echo "* ================================================================ *"
echo "  Configure Instance DBM CFG with Keystore Information"
echo "  ----------------------------------------------------------------"
echo "  db2 update dbm cfg using "
echo "      keystore_type pkcs12"
echo "      keystore_location /home/db2inst1/ccardskeystore.p12"
echo "* ================================================================ *"
echo " "
```

__6. Run script **setup02.sh**

Notice that two DBM CFG parameters have been updated:



```
DB20000I  The UPDATE DATABASE MANAGER CONFIGURATION command completed
successfully.
SQL1362W  One or more of the parameters submitted for immediate modification
were not changed dynamically. Client changes will not be effective until the
next time the application is started or the TERMINATE command has been issued.
Server changes will not be effective until the next DB2START command.

02/13/2015 00:49:27     0   0   SQL1064N  DB2STOP processing was successful.
SQL1064N  DB2STOP processing was successful.
02/13/2015 00:49:29     0   0   SQL1063N  DB2START processing was successful.
SQL1063N  DB2START processing was successful.

 Keystore type                                (KEYSTORE_TYPE) = PKCS12
 Keystore location                        (KEYSTORE_LOCATION) = /home/db2inst1/keystore/ccardskeystore.p12

* ============================================================== *
  End of db2: instance configuration with Keystore info
* ============================================================== *
```

These two DBM CFG parameters are not dynamic. The message says we need to use at least a **TERMINATE** command for the client. But we also must use a **db2stop force** and **db2start** to get these changes to take effect on the server.

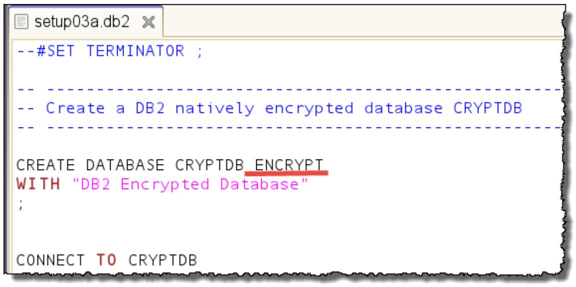
In a DPF environment, the keystore location needs to be accessible to all partitions.

In an IBM pureScale® environment, the keystore location needs to be accessible to all members.

1.5 - Create an encrypted database

Now we can create an encrypted DB2 database using the **ENCRYPT** keyword:

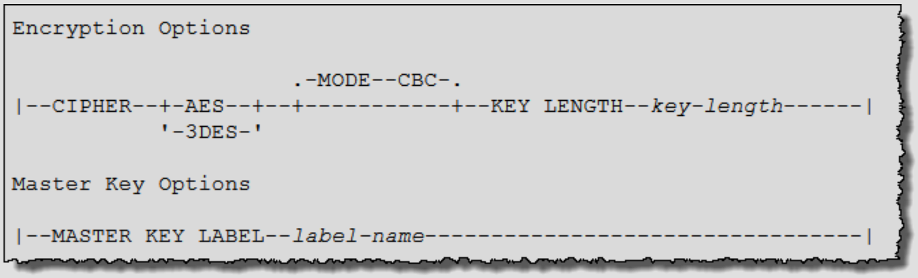
__7. Review script **setup03a.db2.**



```
--#SET TERMINATOR ;

-- ------------------------------------------------------------
-- Create a DB2 natively encrypted database CRYPTDB
-- ------------------------------------------------------------

CREATE DATABASE CRYPTDB ENCRYPT
WITH "DB2 Encrypted Database"
;

CONNECT TO CRYPTDB
```

Note: A master key (and its label) is automatically created during the database creation process and the default encryption was used because we used the simple default keyword: **ENCRYPT**.

You can control encryption further by using other keywords, shown below. The defaults are: **cipher=AES, mode=CBC, key length=256**.



```
Encryption Options

                        .-MODE--CBC-.
|--CIPHER--+-AES--+--+-----------+--KEY LENGTH--key-length------|
           '-3DES-'

Master Key Options

|--MASTER KEY LABEL--label-name---------------------------------|
```

Example: **create db crytdb encrypt cipher 3DES key length 192 master key label mylabel.mydb.myinstance.myserver**

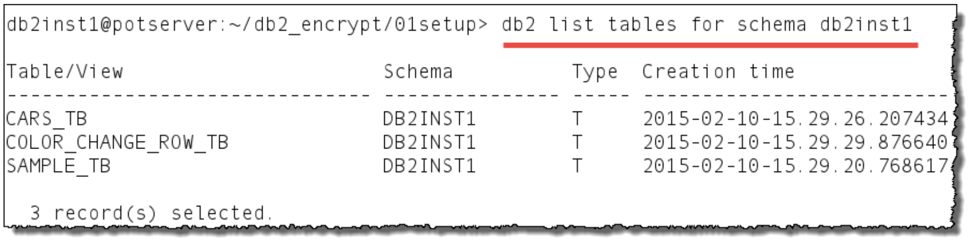
__8. Run script **setup03.sh**

./setup03.sh

__9. When it completes, connect to it and check for the tables created during the script:

db2 connect to cryptdb

db2 list tables for schema db2inst1



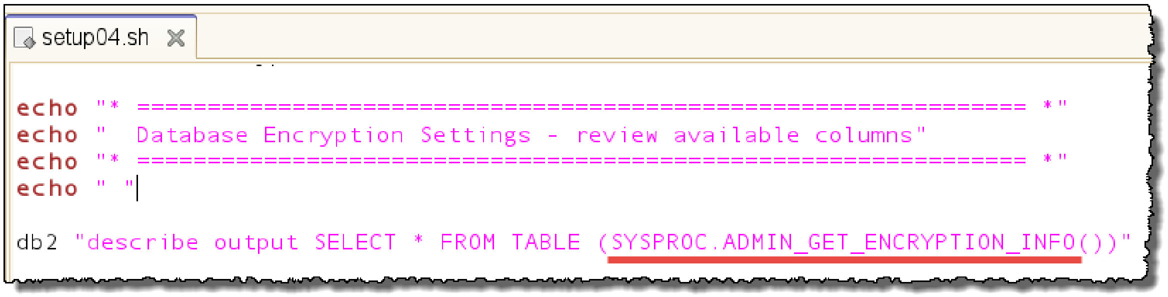
```
db2inst1@potserver:~/db2_encrypt/01setup> db2 list tables for schema db2inst1

Table/View                      Schema          Type  Creation time
------------------------------- --------------- ----- --------------------------
CARS_TB                         DB2INST1        T     2015-02-10-15.29.26.207434
COLOR_CHANGE_ROW_TB             DB2INST1        T     2015-02-10-15.29.29.876640
SAMPLE_TB                       DB2INST1        T     2015-02-10-15.29.20.768617

  3 record(s) selected.
```

1.6 - Check the encryption settings

We can now check what the encryption settings are for our database. This is done using the **`SYSPROC.ADMIN_GET_ENCRYPTION_INFO`** table function.

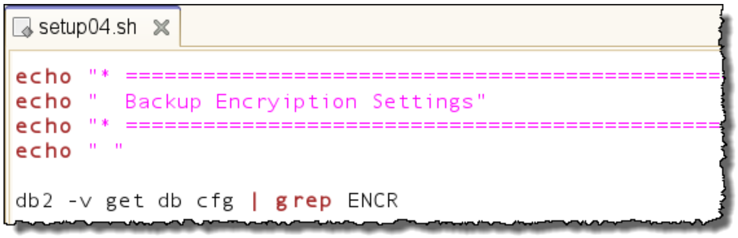
__10. Review script **`setup04.sh`**

Notice that the script uses the table function:



```
echo "* ============================================================= *"
echo "  Database Encryption Settings - review available columns"
echo "* ============================================================= *"
echo " "

db2 "describe output SELECT * FROM TABLE (SYSPROC.ADMIN_GET_ENCRYPTION_INFO())"
```

For the backup encryption settings, it reviews the DB CFG parameters:



```
echo "* ==============================================
echo "  Backup Encryiption Settings"
echo "* ==============================================
echo " "

db2 -v get db cfg | grep ENCR
```

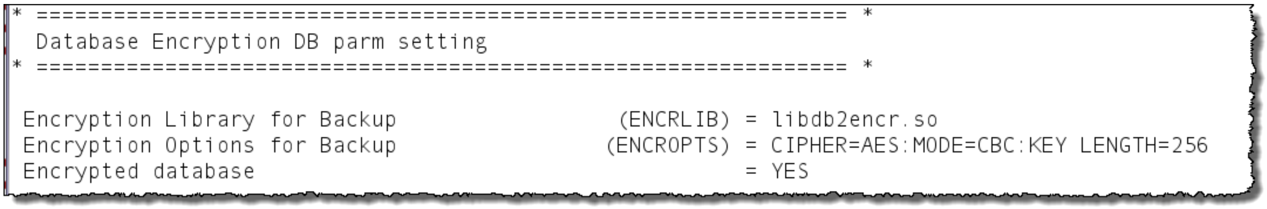
__11. Run and review output from script **`setup04.sh`**

(Tip: expand your terminal window very large so you can to see everything clearly without the output wrapping.)

`./setup04.sh`

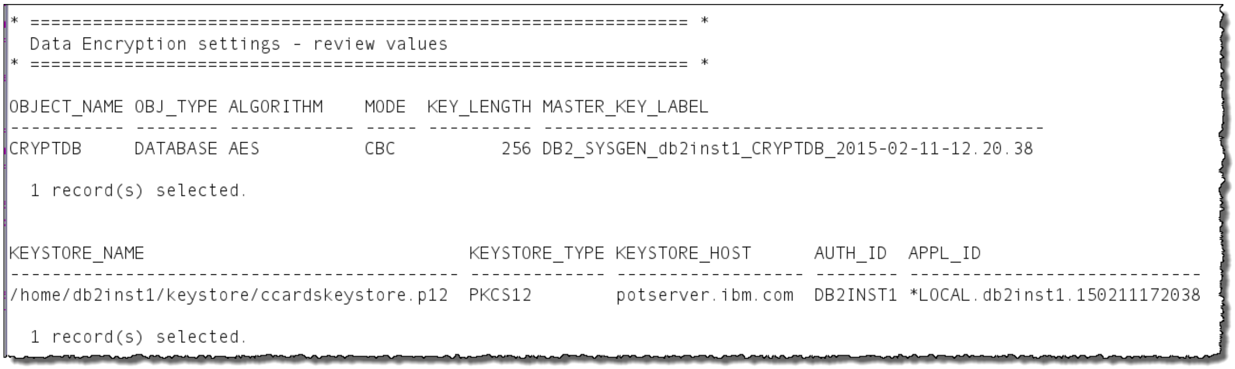
First, notice the DB CFG parm which shows that the database is encrypted (it is not updatable).

Notice also the parameter settings for the backup encryption:



```
* ============================================================= *
  Database Encryption DB parm setting
* ============================================================= *

 Encryption Library for Backup                  (ENCRLIB) = libdb2encr.so
 Encryption Options for Backup                 (ENCROPTS) = CIPHER=AES:MODE=CBC:KEY LENGTH=256
 Encrypted database                                       = YES
```

Finally, notice the output from the table function: **`SYSPROC.ADMIN_GET_ENCRYPTION_INFO`**. The information of note is the MASTER_KEY_LABEL. This points to the master key in the keystore. The keystore is also named, along with its full directory path location.



```
* =============================================================== *
  Data Encryption settings - review values
* =============================================================== *

OBJECT_NAME OBJ_TYPE ALGORITHM    MODE  KEY_LENGTH MASTER_KEY_LABEL
----------- -------- ------------ ----- ---------- ------------------------------------------------
CRYPTDB     DATABASE AES          CBC          256 DB2_SYSGEN_db2inst1_CRYPTDB_2015-02-11-12.20.38

  1 record(s) selected.


KEYSTORE_NAME                               KEYSTORE_TYPE KEYSTORE_HOST      AUTH_ID  APPL_ID
------------------------------------------- ------------- ------------------ -------- ----------------------------
/home/db2inst1/keystore/ccardskeystore.p12  PKCS12        potserver.ibm.com  DB2INST1 *LOCAL.db2inst1.150211172038

  1 record(s) selected.
```

Lab recap

In this lab, we did the following:

- Reviewed the encryption path settings for environment variables
- Created a DB2 keystore (key database)
- Configured the DB2 instance DBM CFG parameters with the Keystore information
- Created the DB2 database with the ENCRYPT keyword
- Reviewed the database and backup encryption settings.

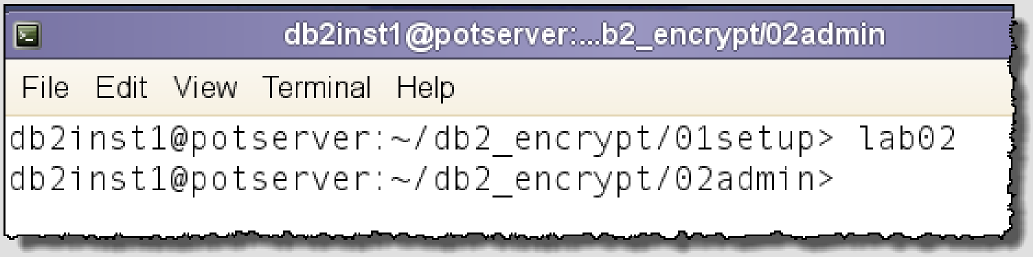
**** End of lab 1: DB2 Native Encryption setup**

Lab 2 - DB2 Native Encryption administration

2.1 - Generate a new master key and master key label

__1. Open the *db2inst1 terminal* (if not already open) and navigate to:
`/home/db2inst1/db2_encrypt/02admin`

Note: To get to this lab directory easily, you can use alias: **`lab02`**

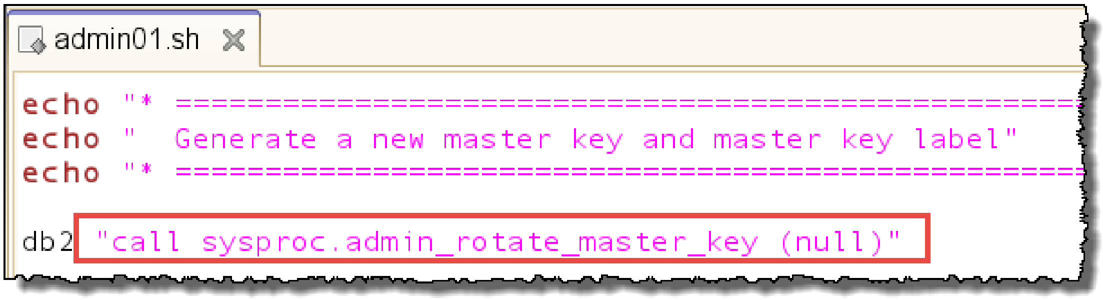

db2inst1@potserver:...b2_encrypt/02admin
File Edit View Terminal Help

```
db2inst1@potserver:~/db2_encrypt/01setup> lab02
db2inst1@potserver:~/db2_encrypt/02admin>
```

Rotation of the master key label should be done according to the organization's security policy. This is similar to changing user passwords at a regular interval, which is also enforced by a security policy.

This is how you can generate a new master key and its label.

__2. Review script **`admin01.sh`**.



```
echo "* ==================================================
echo "  Generate a new master key and master key label"
echo "* ==================================================

db2 "call sysproc.admin_rotate_master_key (null)"
```

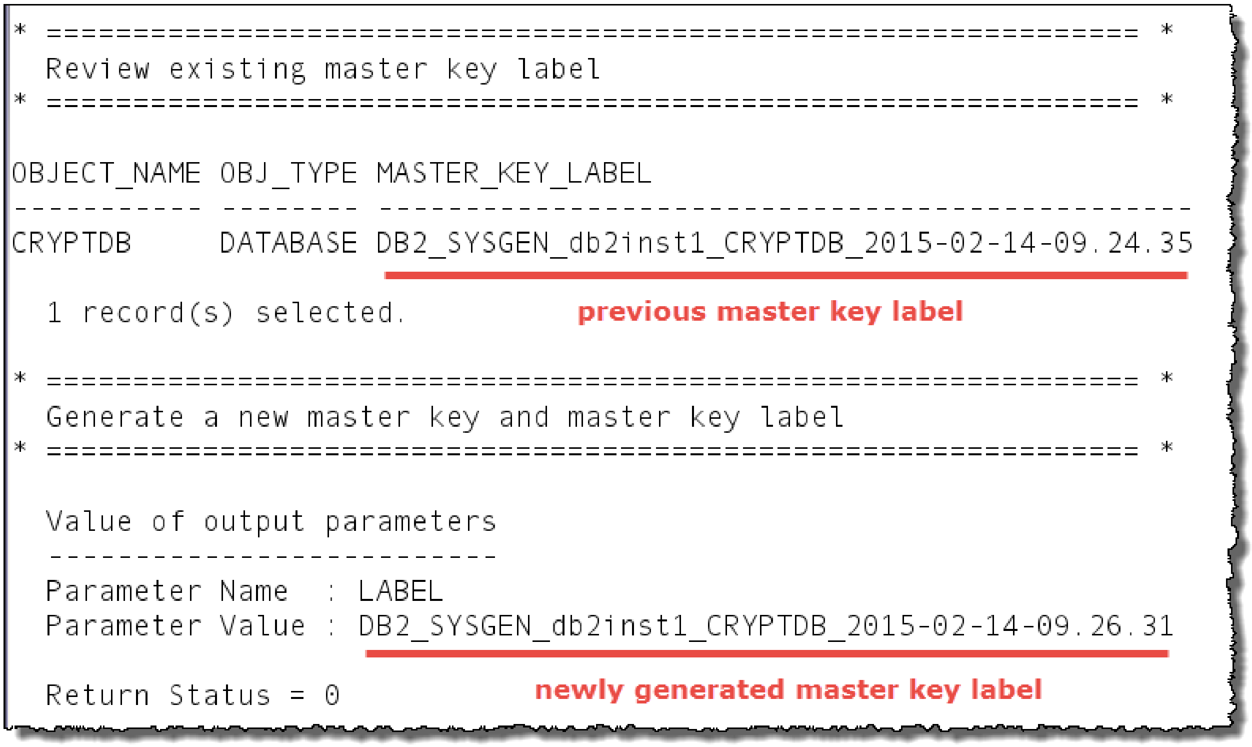
Calling **`sysproc.admin_rotate_master_key`** will a value of **`null`** will generate a new master key and label.

This procedure can also rotate between already existing master keys in the keystore.

__3. Run script **admin01.sh**.

./admin01.sh

Notice the master key label before and after it is changed.



```
* ============================================================== *
  Review existing master key label
* ============================================================== *

OBJECT_NAME OBJ_TYPE MASTER_KEY_LABEL
----------- -------- ----------------------------------------
CRYPTDB     DATABASE DB2_SYSGEN_db2inst1_CRYPTDB_2015-02-14-09.24.35

  1 record(s) selected.

* ============================================================== *
  Generate a new master key and master key label
* ============================================================== *

  Value of output parameters
  --------------------------
  Parameter Name  : LABEL
  Parameter Value : DB2_SYSGEN_db2inst1_CRYPTDB_2015-02-14-09.26.31

  Return Status = 0
```

previous master key label

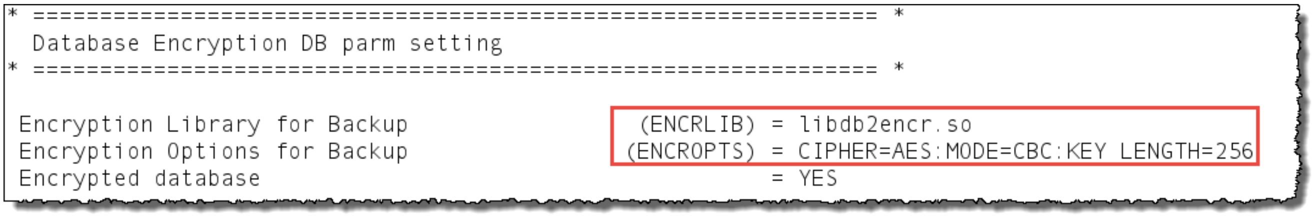
newly generated master key label

The syntax for rotating between existing master keys is as follows:

```
CALL SYSPROC.ADMIN_ROTATE_MASTER_KEY('ExistingLabel')
```

2.2 - Take an encrypted backup

As we saw in a previous exercise, the Backup Encryption Library and Backup Encryption Options were set for us automatically when we created the database with the **ENCRYPT** keyword. See below for what options were assigned by default. (We could have specified different ones, however.)



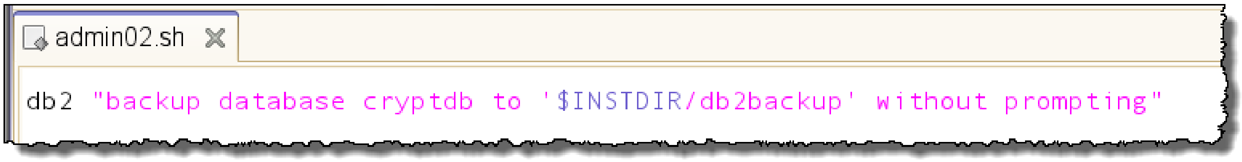
```
* ================================================================ *
  Database Encryption DB parm setting
* ================================================================ *

 Encryption Library for Backup               (ENCRLIB) = libdb2encr.so
 Encryption Options for Backup              (ENCROPTS) = CIPHER=AES:MODE=CBC:KEY LENGTH=256
 Encrypted database                                    = YES
```

DB2 will use these DB CFG parameters when creating backups. Keep in mind that we always have the option of either changing these DB CFG parameters, or specifying an override for them in the **BACKUP** command itself.

__4. Review and run script **admin02.sh**

./admin02.sh

Notice this backup doesn't use any keywords associated with encryption. It will pick up the encryption library and options from the DB CFG parameters.

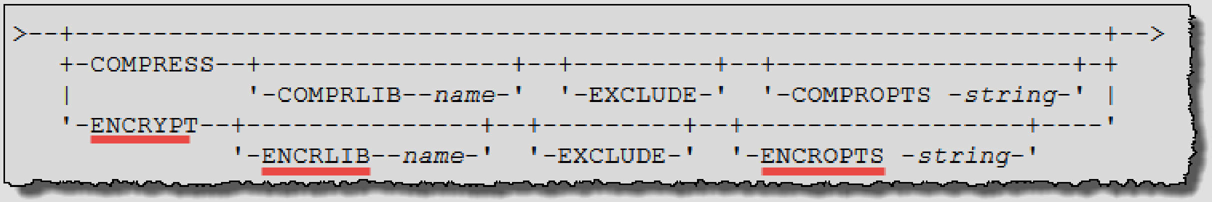


```
db2 "backup database cryptdb to '$INSTDIR/db2backup' without prompting"
```

The **BACKUP** command can provide encryption parameters for two reasons:

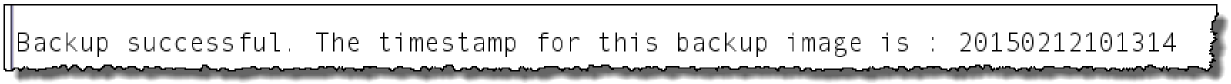
1. If you want to override the DB CFG parameters in an encrypted database
2. If you are not using an encrypted database but wish to encrypt the backup on its own

The **BACKUP** encryption syntax looks like this:



```
>--+-----------------------------------------------------------------+-->
   +-COMPRESS--+-----------------+--+----------+--+-------------------+-+
   |           '-COMPRLIB--name-'  '-EXCLUDE-'  '-COMPROPTS -string-' |
   '-ENCRYPT--+----------------+--+----------+--+-------------------+----'
               '-ENCRLIB--name-'  '-EXCLUDE-'  '-ENCROPTS -string-'
```

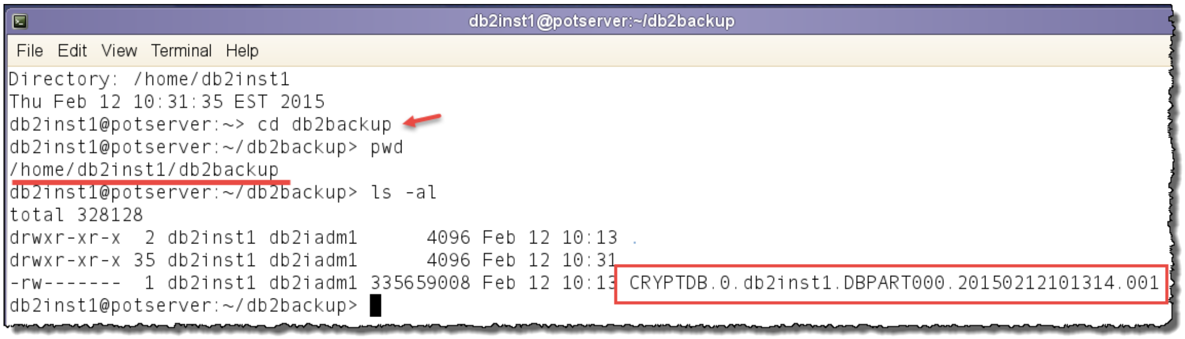
__5. When the script completes, you will get a “backup successful” message.



```
Backup successful. The timestamp for this backup image is : 20150212101314
```

__6. Open a second terminal window and change directories to the db2backup directory to find the new backup image:

```
cd db2backup
ls -al
```

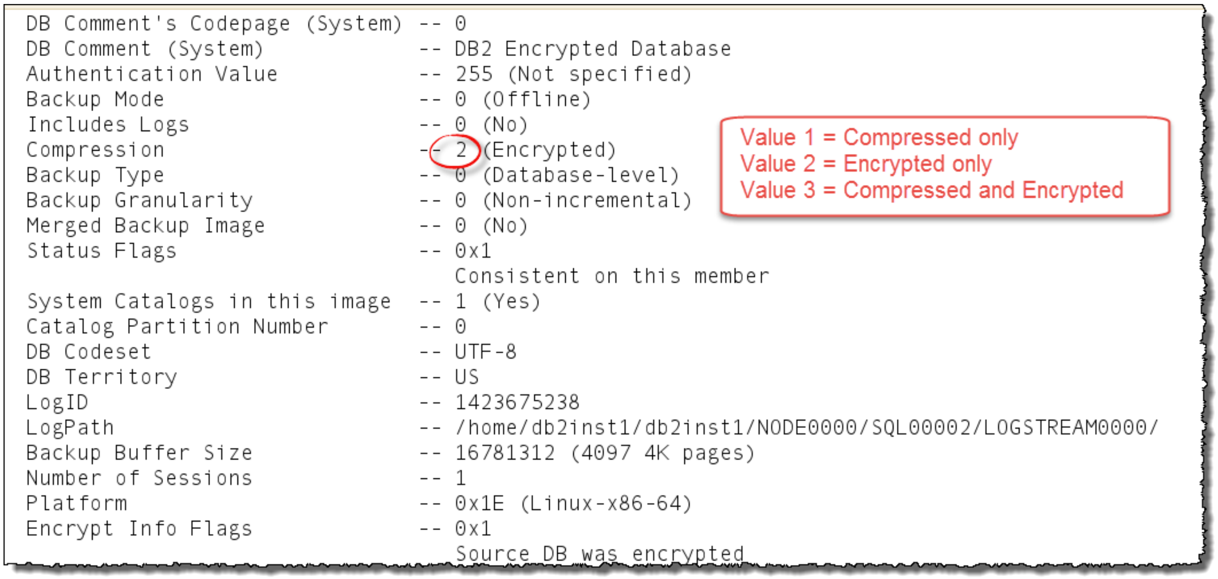


```
Directory: /home/db2inst1
Thu Feb 12 10:31:35 EST 2015
db2inst1@potserver:~> cd db2backup
db2inst1@potserver:~/db2backup> pwd
/home/db2inst1/db2backup
db2inst1@potserver:~/db2backup> ls -al
total 328128
drwxr-xr-x  2 db2inst1 db2iadm1      4096 Feb 12 10:13 .
drwxr-xr-x 35 db2inst1 db2iadm1      4096 Feb 12 10:31 ..
-rw-------  1 db2inst1 db2iadm1 335659008 Feb 12 10:13 CRYPTDB.0.db2inst1.DBPART000.20150212101314.001
db2inst1@potserver:~/db2backup> 
```

__7. Run a check of the backup to make sure it is encrypted. An example looks like this:

```
db2ckbkp -H CRYPTDB.0.db2inst1.DBPART000.20150212101314.001
```

(Your backup image name will be different from the above example because a timestamp is part of the backup file naming convention. Use the file name you see in your own directory.)



```
 DB Comment's Codepage (System) -- 0
 DB Comment (System)            -- DB2 Encrypted Database
 Authentication Value           -- 255 (Not specified)
 Backup Mode                    -- 0 (Offline)
 Includes Logs                  -- 0 (No)
 Compression                    -- 2 (Encrypted)
 Backup Type                    -- 0 (Database-level)
 Backup Granularity             -- 0 (Non-incremental)
 Merged Backup Image            -- 0 (No)
 Status Flags                   -- 0x1
                                   Consistent on this member
 System Catalogs in this image  -- 1 (Yes)
 Catalog Partition Number       -- 0
 DB Codeset                     -- UTF-8
 DB Territory                   -- US
 LogID                          -- 1423675238
 LogPath                        -- /home/db2inst1/db2inst1/NODE0000/SQL00002/LOGSTREAM0000/
 Backup Buffer Size             -- 16781312 (4097 4K pages)
 Number of Sessions             -- 1
 Platform                       -- 0x1E (Linux-x86-64)
 Encrypt Info Flags             -- 0x1
                                   Source DB was encrypted
```

Value 1 = Compressed only
Value 2 = Encrypted only
Value 3 = Compressed and Encrypted

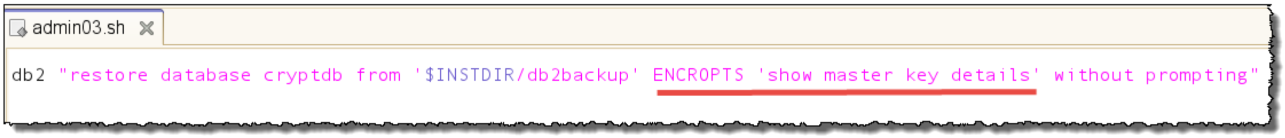
This backup image header dump shows the backup to be encrypted.

2.3 - Review master key information from a backup image

If you are not certain what master key information you used during a previous backup, you can always dump that information from the backup image itself during a **RESTORE** or **RECOVER** operation. Simply use these keywords during those operations: **ENCROPTS 'show master key details'**

__8. Review script **admin03.sh**

Notice the use of the keywords that dump the encryption master key information:

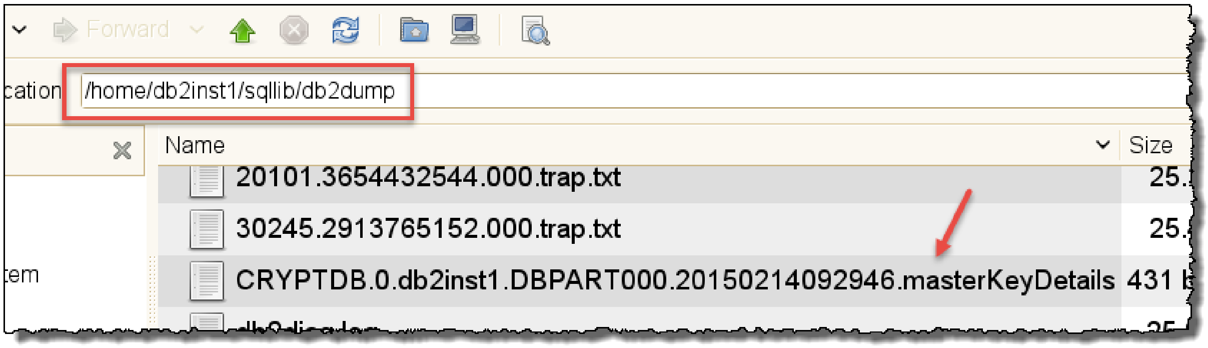


```
db2 "restore database cryptdb from '$INSTDIR/db2backup' ENCROPTS 'show master key details' without prompting"
```

__9. Run script **admin03.sh**

./admin03.sh

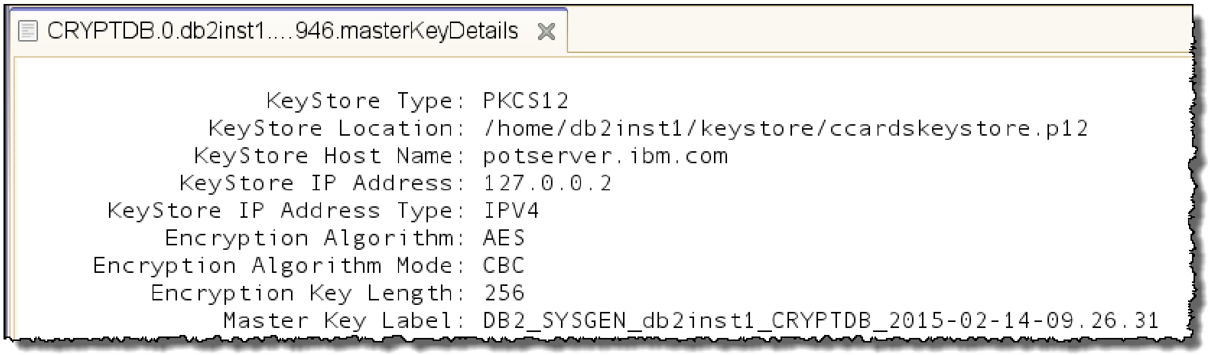
When it finishes the **RESTORE**, use the file browser (or **cd** and **ls** commands) to find the **masterKeyDetails** file in the instance dump directory: **/home/db2inst1/sqllib/db2dump.**



/home/db2inst1/sqllib/db2dump

Name	Size
20101.3654432544.000.trap.txt	25.
30245.2913765152.000.trap.txt	25.
CRYPTDB.0.db2inst1.DBPART000.20150214092946.masterKeyDetails	431 b

__10. Browse the file for details:



```
             KeyStore Type: PKCS12
         KeyStore Location: /home/db2inst1/keystore/ccardskeystore.p12
        KeyStore Host Name: potserver.ibm.com
       KeyStore IP Address: 127.0.0.2
  KeyStore IP Address Type: IPV4
      Encryption Algorithm: AES
 Encryption Algorithm Mode: CBC
     Encryption Key Length: 256
          Master Key Label: DB2_SYSGEN_db2inst1_CRYPTDB_2015-02-14-09.26.31
```

2.4 - Generate and add a new master key

In the event that you want to generate and add your own master key, this is how you do it.

__11. Review and run script **`admin04.sh`**

```
./admin04.sh
```

Review the version and help information that is returned from this script.

2.5 - Review various GSKit utility commands

The GSKit utility **`gsk8capicmd`** is the 32-bit version. Our VM image uses the 64-bit version called **`gsk8capicmd_64`**. This exercise will review a few of the commands you might need to use in the future when administering native encryption in DB2.

__12. Review and run script **`admin05.sh`**

```
./admin05.sh
```

Review the version and help information that is returned from this script.

Lab recap

In this lab, we did the following:

- Generated a master key and master key label
- Took an encrypted backup
- Checked the backup header to make sure it is encrypted
- Reviewed the master key detailed information from a backup image
- Explored various GSKit utility commands

**** End of lab 2: DB2 Native Encryption Administration**

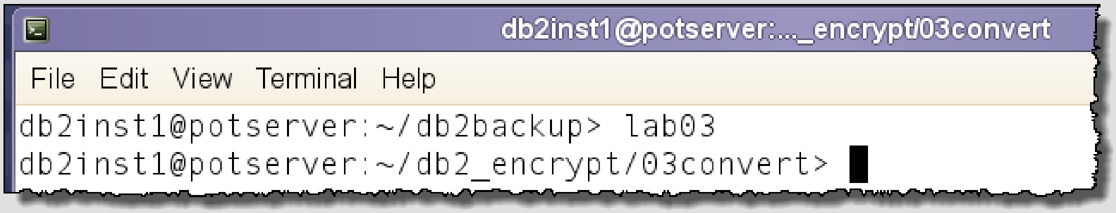
Lab 3 - Convert a cleartext database to ciphertext (encrypted)

3.1 - Create a cleartext (non-encrypted) database

__1. Open the *db2inst1 terminal* (if not already open) and navigate to:

`/home/db2inst1/db2_encrypt/03convert`

Note: To get to this lab directory easily, you can use alias: **`lab03`**


db2inst1@potserver:..._encrypt/03convert
File Edit View Terminal Help

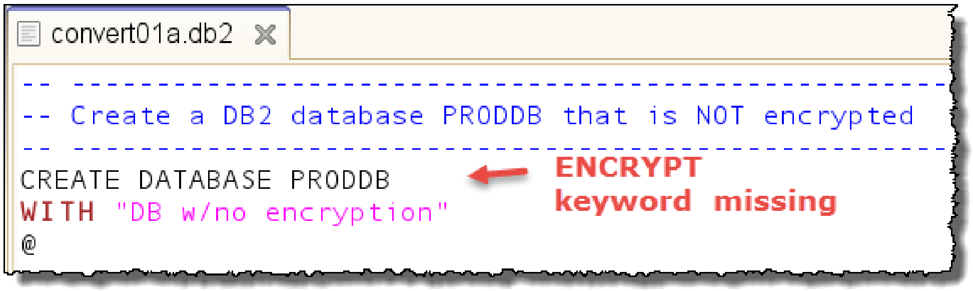
```
db2inst1@potserver:~/db2backup> lab03
db2inst1@potserver:~/db2_encrypt/03convert>
```

There is no method to encrypt a cleartext DB2 database "in place." You have to perform a backup of the database you wish to encrypt first, and then restore it with the encrypt command to accomplish this task.

We will create a cleartext DB2 database to simulate the scenario you might have in your shop of encrypting an existing database.

__2. Review script **`convert01a.db2`**.

Notice database PRODDB will be created as a cleartext database because we will not use the keyword **`ENCRYPT`**.



```
-- -----------------------------------------------------
-- Create a DB2 database PRODDB that is NOT encrypted
-- -----------------------------------------------------
CREATE DATABASE PRODDB
WITH "DB w/no encryption"
@
```

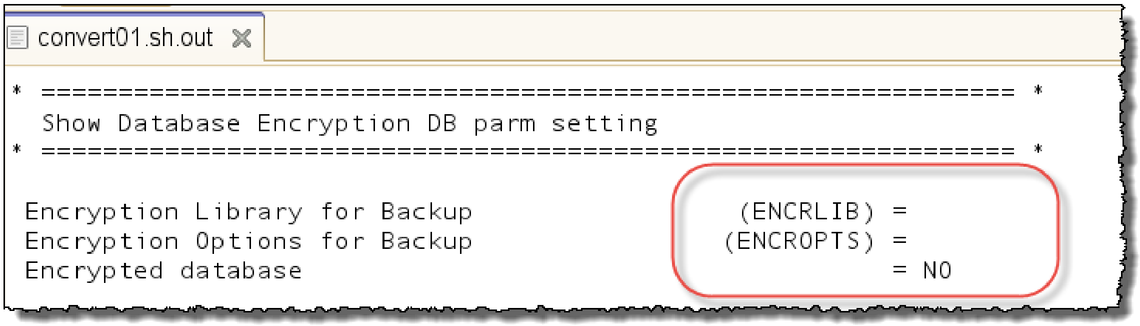
ENCRYPT keyword missing

__3. Run script **`convert01.sh`**.

`./convert01.sh`

__4. After it is completed, review the output file **`convert01.sh.out`**

This output from the database creation shows what the Encryption DB CFG parameters look like for a cleartext database:

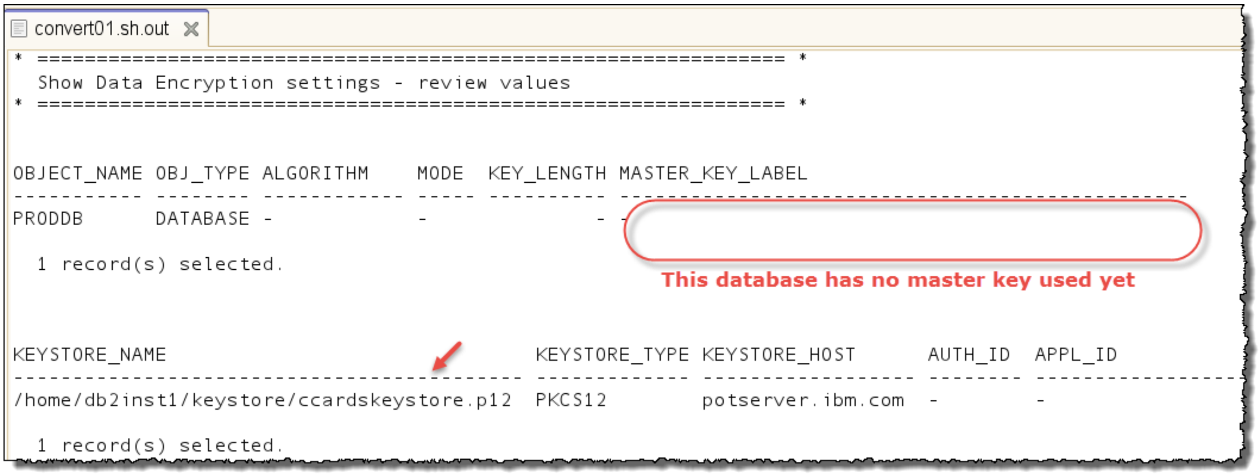


```
* ================================================================ *
  Show Database Encryption DB parm setting
* ================================================================ *

 Encryption Library for Backup                 (ENCRLIB) =
 Encryption Options for Backup                (ENCROPTS) =
 Encrypted database                                      = NO
```

Reviewing the output from the table function **`sysproc.admin_get_encryption_info`**, you should notice that there are some encryption values set for this database.

For example, the keystore is set by means of the DBM CFG parameter (but the database is not using it). We know that the database is not encrypted because of the DB CFG parameter above that says so specifically.

We also can tell the database is not encrypted because there is no master key label for the database either.



```
* ================================================================ *
  Show Data Encryption settings - review values
* ================================================================ *


OBJECT_NAME OBJ_TYPE ALGORITHM    MODE  KEY_LENGTH MASTER_KEY_LABEL
----------- -------- ------------ ----- ---------- --------------------------------------------------
PRODDB      DATABASE -            -              - -

  1 record(s) selected.


KEYSTORE_NAME                                      KEYSTORE_TYPE KEYSTORE_HOST      AUTH_ID  APPL_ID
-------------------------------------------------- ------------- ------------------ -------- ------------------
/home/db2inst1/keystore/ccardskeystore.p12         PKCS12        potserver.ibm.com  -        -

  1 record(s) selected.
```

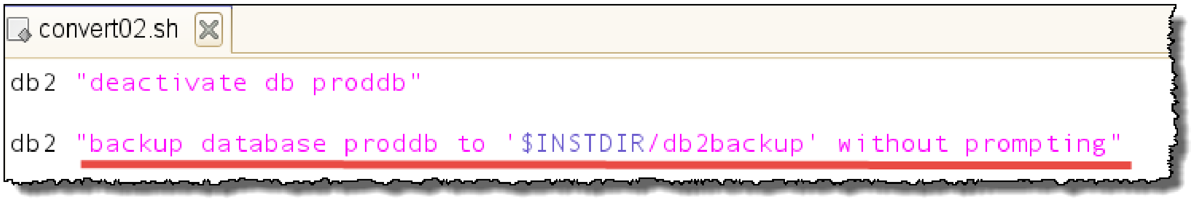
This database has no master key used yet

3.2 - Make a backup of the cleartext database

Now we will create a backup of the PRODDB database so we can encrypt it during a restore.

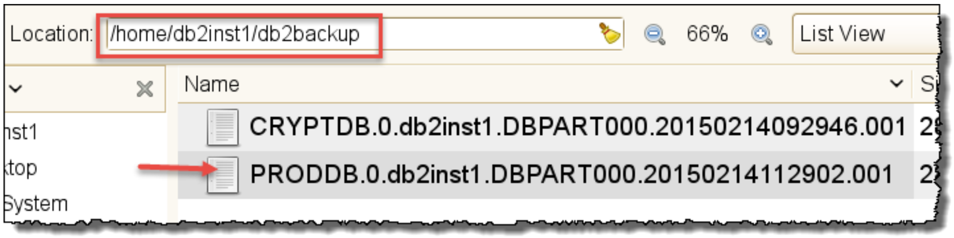
__5. Review and run script **`convert02.sh`**.

This will take a full offline backup of the PRODDB database:



```
db2 "deactivate db proddb"

db2 "backup database proddb to '$INSTDIR/db2backup' without prompting"
```

__6. Check the backup directory to make sure the backup is there. It is located in this directory: **`/home/db2inst1/db2backup`**

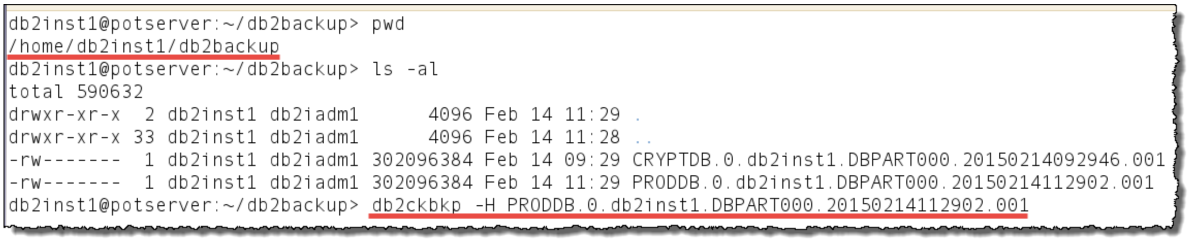


/home/db2inst1/db2backup

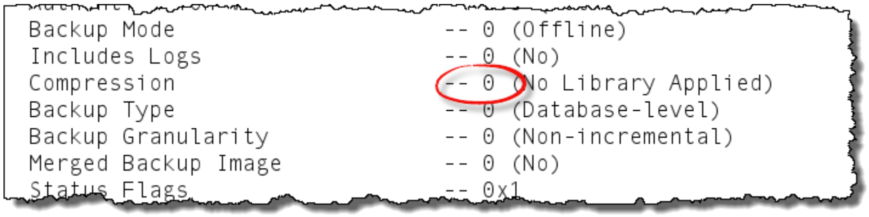
CRYPTDB.0.db2inst1.DBPART000.20150214092946.001

PRODDB.0.db2inst1.DBPART000.20150214112902.001

__7. Check the header of that backup to prove to yourself that the backup image itself is not encrypted:



```
db2inst1@potserver:~/db2backup> pwd
/home/db2inst1/db2backup
db2inst1@potserver:~/db2backup> ls -al
total 590632
drwxr-xr-x  2 db2inst1 db2iadm1      4096 Feb 14 11:29 .
drwxr-xr-x 33 db2inst1 db2iadm1      4096 Feb 14 11:28 ..
-rw-------  1 db2inst1 db2iadm1 302096384 Feb 14 09:29 CRYPTDB.0.db2inst1.DBPART000.20150214092946.001
-rw-------  1 db2inst1 db2iadm1 302096384 Feb 14 11:29 PRODDB.0.db2inst1.DBPART000.20150214112902.001
db2inst1@potserver:~/db2backup> db2ckbkp -H PRODDB.0.db2inst1.DBPART000.20150214112902.001
```



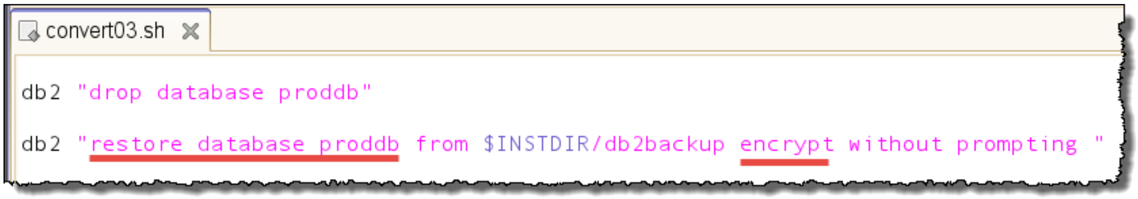
```
 Backup Mode                           -- 0 (Offline)
 Includes Logs                         -- 0 (No)
 Compression                           -- 0 (No Library Applied)
 Backup Type                           -- 0 (Database-level)
 Backup Granularity                    -- 0 (Non-incremental)
 Merged Backup Image                   -- 0 (No)
 Status Flags                          -- 0x1
```

3.3 - Restore the cleartext database with ENCRYPT

Now we will restore PRODDB with the **ENCRYPT** keyword in order to convert it from cleartext to ciphertext.

__8. Review **convert03.sh**.

Notice that this script will restore the PRODDB database, only now it will use the **ENCRYPT** keyword to encrypt the database during the restore operation.



```
db2 "drop database proddb"

db2 "restore database proddb from $INSTDIR/db2backup encrypt without prompting "
```

We had to drop the database PRODDB rather than **RESTORE** over it. This is because it is not possible to override the database setting that tells whether it is using encryption. Since the cleartext PRODDB had **Encrypt=NO**, we had to drop the database in order to **RESTORE** it with encryption.

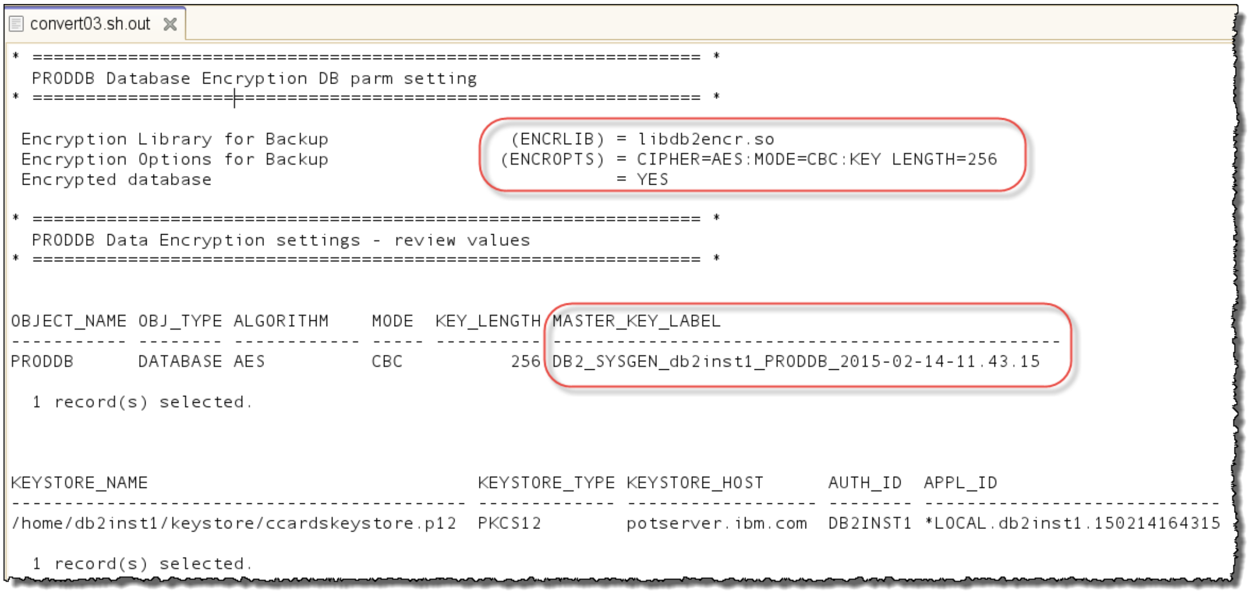
Because the encryption DBM CFG parameters are set for the instance, **RESTORE** has the same effect as **CREATE** when using the **ENCRYPT** parameter. It has the added value of encrypting all the data in the backup image as it is being restored.

__9. Run script **convert03.sh** to perform the restore.

./convert03.sh

__10. Review the output file from this script: **`convert03.sh.out`**

Notice now that the database is encrypted and the DB CFG parameters for the backups are set. Also notice that there is a master key label associated with the database just as it was for the database CRYPTDB that we created new with the **`ENCRYPT`** keyword.



```
* ============================================================== *
  PRODDB Database Encryption DB parm setting
* ============================================================== *

 Encryption Library for Backup              (ENCRLIB) = libdb2encr.so
 Encryption Options for Backup             (ENCROPTS) = CIPHER=AES:MODE=CBC:KEY LENGTH=256
 Encrypted database                                   = YES

* ============================================================== *
  PRODDB Data Encryption settings - review values
* ============================================================== *


OBJECT_NAME OBJ_TYPE ALGORITHM    MODE  KEY_LENGTH MASTER_KEY_LABEL
----------- -------- ------------ ----- ---------- ----------------------------------------------
PRODDB      DATABASE AES          CBC          256 DB2_SYSGEN_db2inst1_PRODDB_2015-02-14-11.43.15

  1 record(s) selected.


KEYSTORE_NAME                               KEYSTORE_TYPE KEYSTORE_HOST     AUTH_ID  APPL_ID
------------------------------------------- ------------- ----------------- -------- ----------------------------
/home/db2inst1/keystore/ccardskeystore.p12  PKCS12        potserver.ibm.com DB2INST1 *LOCAL.db2inst1.150214164315

  1 record(s) selected.
```

If you are using archive logging and have taken an online backup, you can still **`RESTORE`** with a **`ROLLFORWARD`** or **`RECOVER`** the database and use the keyword **`ENCRYPT`**.

If you do this, the newly restored or recovered database will be encrypted in the same way as we just accomplished with an offline **`BACKUP`** and **`RESTORE`**. The only difference is that archive logs will be read in the process.

Lab recap

In this lab, we accomplished the following:

- Create a cleartext (non encrypted) database
- Back up the cleartext database
- Restore the database using ENCRYPT

**** End of lab 3: Convert a cleartext database to ciphertext (encrypted)**

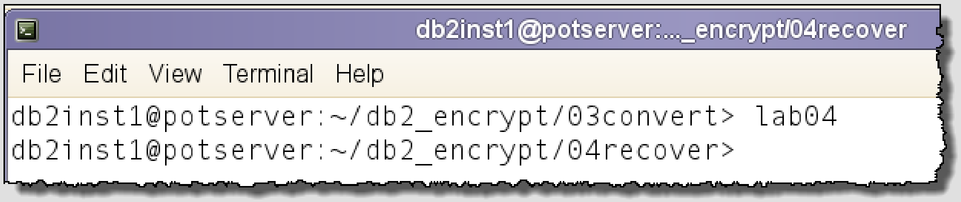
Lab 4 - Keystore recovery scenarios

4.1 - Losing a keystore stash file

__1. Open the *db2inst1 terminal* (if not already open) and navigate to:

`/home/db2inst1/db2_encrypt/04recover`

Note: To get to this lab directory easily, you can use alias: **`lab04`**

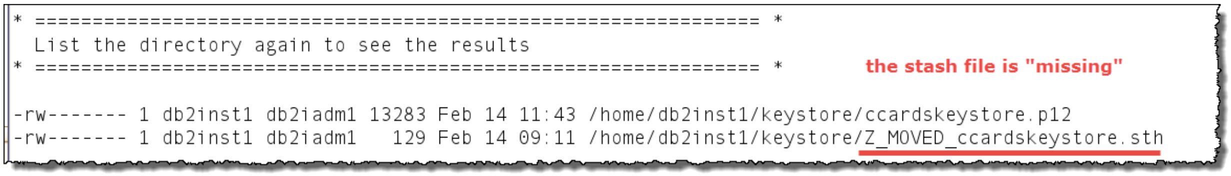

db2inst1@potserver:..._encrypt/04recover
File Edit View Terminal Help

```
db2inst1@potserver:~/db2_encrypt/03convert> lab04
db2inst1@potserver:~/db2_encrypt/04recover>
```

If you have been using a keystore hash file to automatically authenticate to the keystore during any access to your encrypted database, what happens when you lose it?

__2. Review and run script **`recover01.sh`**.

The script activates database CRYPTDB and then "loses" the stash file by renaming it.

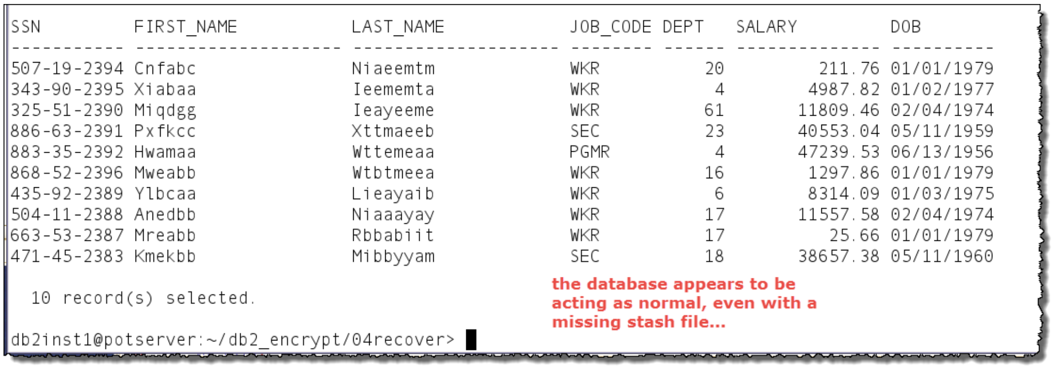


```
* ============================================================= *
  List the directory again to see the results
* ============================================================= *

-rw------- 1 db2inst1 db2iadm1 13283 Feb 14 11:43 /home/db2inst1/keystore/ccardskeystore.p12
-rw------- 1 db2inst1 db2iadm1   129 Feb 14 09:11 /home/db2inst1/keystore/Z_MOVED_ccardskeystore.sth
```

the stash file is "missing"

__3. Now try these commands to see if this affects our active database:

```
db2 connect to cryptdb
db2 list tables for schema db2inst1
db2 "select * from sample_tb fetch first 10 rows only"
```

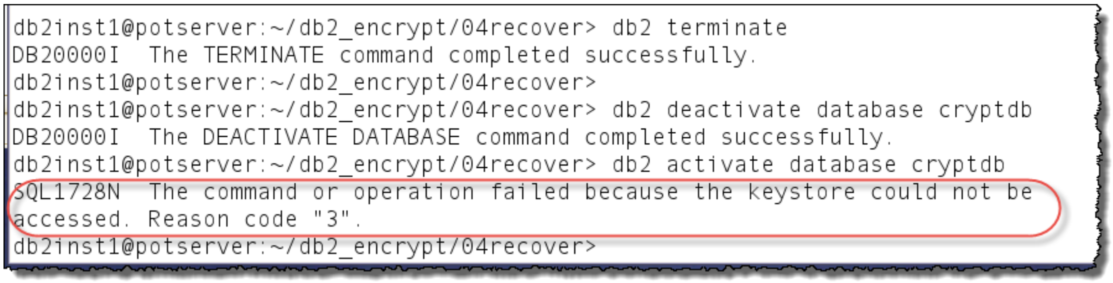


```
SSN         FIRST_NAME           LAST_NAME            JOB_CODE DEPT   SALARY         DOB
----------- -------------------- -------------------- -------- ------ -------------- ----------
507-19-2394 Cnfabc               Niaeemtm             WKR          20         211.76 01/01/1979
343-90-2395 Xiabaa               Ieememta             WKR           4        4987.82 01/02/1977
325-51-2390 Miqdgg               Ieayeeme             WKR          61       11809.46 02/04/1974
886-63-2391 Pxfkcc               Xttmaeeb             SEC          23       40553.04 05/11/1959
883-35-2392 Hwamaa               Wttemeaa             PGMR          4       47239.53 06/13/1956
868-52-2396 Mweabb               Wtbtmeea             WKR          16        1297.86 01/01/1979
435-92-2389 Ylbcaa               Lieayaib             WKR           6        8314.09 01/03/1975
504-11-2388 Anedbb               Niaaayay             WKR          17       11557.58 02/04/1974
663-53-2387 Mreabb               Rbbabiit             WKR          17          25.66 01/01/1979
471-45-2383 Kmekbb               Mibbyyam             SEC          18       38657.38 05/11/1960

  10 record(s) selected.

db2inst1@potserver:~/db2_encrypt/04recover>
```

the database appears to be acting as normal, even with a missing stash file...

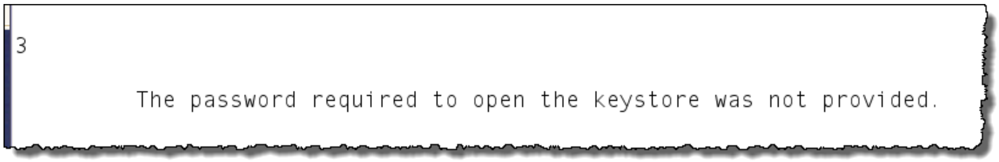
__4. Now try these commands to see what happens when you re-activate a database in this situation:

```
db2 terminate
db2 deactivate database cryptdb
db2 activate database cryptdb
```



```
db2inst1@potserver:~/db2_encrypt/04recover> db2 terminate
DB20000I  The TERMINATE command completed successfully.
db2inst1@potserver:~/db2_encrypt/04recover>
db2inst1@potserver:~/db2_encrypt/04recover> db2 deactivate database cryptdb
DB20000I  The DEACTIVATE DATABASE command completed successfully.
db2inst1@potserver:~/db2_encrypt/04recover> db2 activate database cryptdb
SQL1728N  The command or operation failed because the keystore could not be
accessed. Reason code "3".
db2inst1@potserver:~/db2_encrypt/04recover>
```

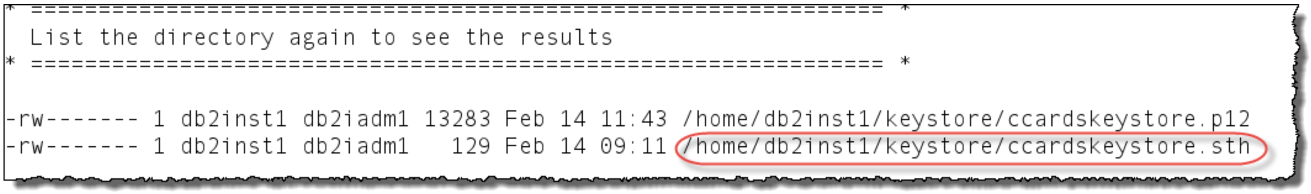
Notice the reason code "3" message we receive:



```
3

         The password required to open the keystore was not provided.
```

__5. Review and run script **`recover02.sh`**. (This will rename the stash file, "recovering" it back to its location again which simulates "recovering" this file from a backup you have taken previously.)

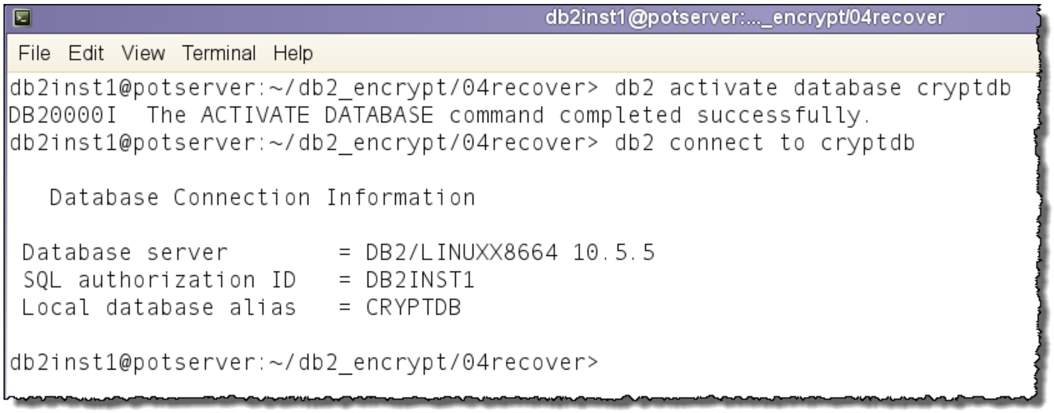
```
./recover02.sh
```



```
* ================================================================ *
  List the directory again to see the results
* ================================================================ *

-rw------- 1 db2inst1 db2iadm1 13283 Feb 14 11:43 /home/db2inst1/keystore/ccardskeystore.p12
-rw------- 1 db2inst1 db2iadm1   129 Feb 14 09:11 /home/db2inst1/keystore/ccardskeystore.sth
```

__6. Try using the database again with these commands:

```
db2 activate database cryptdb
db2 connect to cryptdb
```


db2inst1@potserver:..._encrypt/04recover
File Edit View Terminal Help

```
db2inst1@potserver:~/db2_encrypt/04recover> db2 activate database cryptdb
DB20000I  The ACTIVATE DATABASE command completed successfully.
db2inst1@potserver:~/db2_encrypt/04recover> db2 connect to cryptdb

   Database Connection Information

 Database server        = DB2/LINUXX8664 10.5.5
 SQL authorization ID   = DB2INST1
 Local database alias   = CRYPTDB

db2inst1@potserver:~/db2_encrypt/04recover>
```

4.2 - Starting an instance without a keystore stash file

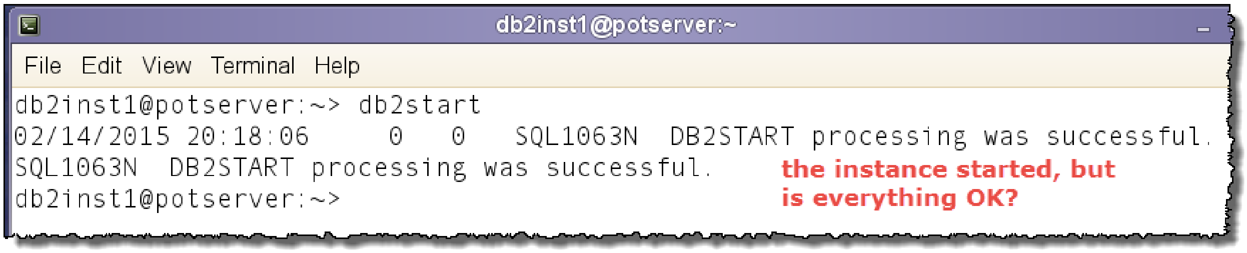
You don't have to use a stash file with DB2 Native Encryption. In this exercise, we will start the DB2 instance without one.

__7. Review and run script **`recover03.sh`**.

```
./recover03.sh
```

__8. Now start the DB2 instance:

```
db2start
```

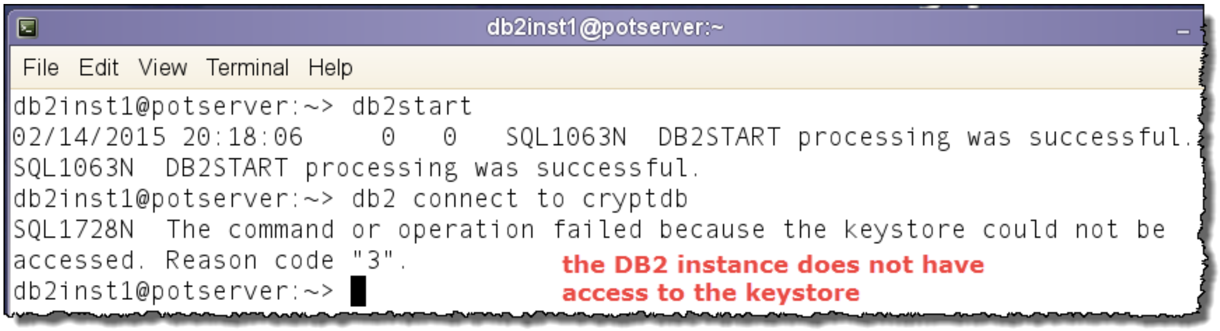


```
db2inst1@potserver:~> db2start
02/14/2015 20:18:06     0   0   SQL1063N  DB2START processing was successful.
SQL1063N  DB2START processing was successful.
db2inst1@potserver:~>
```

the instance started, but is everything OK?

__9. Everything appears to be normal. But now, try to connect to database CRYPTDB:

```
db2 connect to cryptdb
```



```
db2inst1@potserver:~> db2start
02/14/2015 20:18:06     0   0   SQL1063N  DB2START processing was successful.
SQL1063N  DB2START processing was successful.
db2inst1@potserver:~> db2 connect to cryptdb
SQL1728N  The command or operation failed because the keystore could not be
accessed. Reason code "3".
db2inst1@potserver:~>
```

the DB2 instance does not have access to the keystore

The instance was not able to access the keystore because we don't have a stash file that provides the password to the keystore. As a result, the database could not activate.

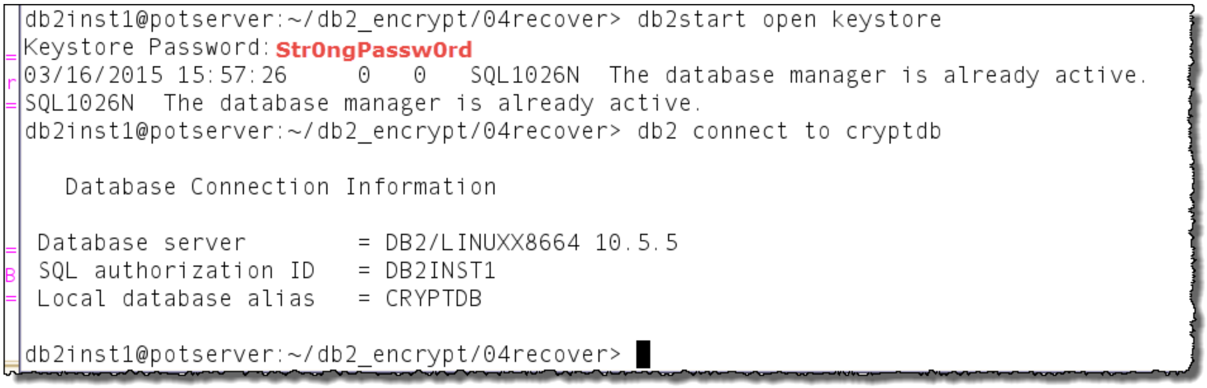
__10. You can issue this command to provide a password to the keystore without stopping the instance:

```
db2start open keystore
```

__11. When prompted for the keystore password, enter the one that we used in our keystore creation script back in the first lab exercise: **`Str0ngPassw0rd`**

Note: This password has capital letters, as well as zeroes in it (not the letter O).

After the instance starts, you should be able to connect to the CRYPTDB database as normal.



```
db2inst1@potserver:~/db2_encrypt/04recover> db2start open keystore
Keystore Password: Str0ngPassw0rd
03/16/2015 15:57:26     0   0   SQL1026N  The database manager is already active.
SQL1026N  The database manager is already active.
db2inst1@potserver:~/db2_encrypt/04recover> db2 connect to cryptdb

   Database Connection Information

 Database server        = DB2/LINUXX8664 10.5.5
 SQL authorization ID   = DB2INST1
 Local database alias   = CRYPTDB

db2inst1@potserver:~/db2_encrypt/04recover> 
```

You could rename the stash file again by rerunning: **`recover02.sh`**. Then, stop and start the instance normally. Connect to database CRYPTDB.

--

FINAL NOTE: Always back up your keystore and stash files using a secure copy protocol (like SCP) to another server's protected area. If you lose either, you can replace them and get on with using your database again, just like in these exercises.

Lab recap

These are the recover exercises we did in this lab:

- Simulated losing a stash file and trying to connect to a database, then recovering from that
- Started a DB2 instance without having a stash file

**** End of lab 4: Keystore recovery scenarios.**

Appendix A. 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 grant you any license to these patents. You can send license inquiries, in writing, to:

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

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

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

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

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

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

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

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

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

All statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only.

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. All references to fictitious companies or individuals are used for illustration purposes only.

Appendix B. Trademarks and copyrights

The following terms are trademarks of International Business Machines Corporation in the United States, other countries, or both:

IBM	AIX	CICS	ClearCase	ClearQuest	Cloudscape
Cube Views	DB2	developerWorks	DRDA	IMS	IMS/ESA
Informix	Lotus	Lotus Workflow	MQSeries	OmniFind	
Rational	Redbooks	Red Brick	RequisitePro	System i	
System z	*Tivoli*	*WebSphere*	*Workplace*	*System p*	

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

IT Infrastructure Library is a registered trademark of the Central Computer and Telecommunications Agency which is now part of the Office of Government Commerce.

Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

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

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

ITIL is a registered trademark, and a registered community trademark of The Minister for the Cabinet Office, and is registered in the U.S. Patent and Trademark Office.

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

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Cell Broadband Engine is a trademark of Sony Computer Entertainment, Inc. in the United States, other countries, or both and is used under license therefrom.

Linear Tape-Open, LTO, the LTO Logo, Ultrium, and the Ultrium logo are trademarks of HP, IBM Corp. and Quantum in the U.S. and other countries.

NOTES

NOTES