Oracle Database

Integration Guide

gemalto
security to be free

Document Part Number: 007-008670-001, Rev. AN

Release Date: September 2016

Contents

Preface

This document is intended to guide security administrators through the steps for the Oracle Database 11g and 12c Integration with SafeNet Network HSM / SafeNet PCIe HSM, and also covers the necessary information to install, configure and integrate Oracle Database Transparent Data Encryption (TDE) with SafeNet Network HSM / SafeNet PCIe HSM.

Scope

This document outlines the steps to integrate Oracle Database with SafeNet HSM. SafeNet HSM is used to secure the Master Encryption Key for Oracle TDE in FIPS 140-2 Approved HSM.

While multiple applications per partition is supported by the PKCS #11 standard, it is the customer's responsibility to validate that applications sharing a partition do so in a manner that does not result in conflicts between the applications.

Gemalto/SafeNet strongly recommends customers implement frequent, high quality backups of the key material held within the HSM in a manner consistent with the value of the data being protected by them. It is critical to align this backup strategy to with the application's key management behavior. For example, backups must be refreshed immediately after the application performs any key rotation activities; backups should also be refreshed before any application or HSM upgrades are performed. This is particularly critical for data encryption applications, where loss of the keys renders the data inaccessible.

Backups should be verified on a regular basis both to ensure they are usable and to ensure the team responsible for the system knows how to use them to perform a recovery.

SafeNet also recommends users thoroughly test all application and HSM upgrades before deploying them on production systems. Depending on the customer's deployments behavior changes in the application or HSM may have an unexpected result.

Gemalto Rebranding

In early 2015, Gemalto completed its acquisition of SafeNet, Inc. As part of the process of rationalizing the product portfolios between the two organizations, the Luna name has been removed from the SafeNet HSM product line, with the SafeNet name being retained. As a result, the product names for SafeNet HSMs have changed as follows:

Old product name	New product name
Luna SA HSM	SafeNet Network HSM
Luna PCI-E HSM	SafeNet PCI-E HSM
Luna G5 HSM	SafeNet USB HSM
Luna Client	SafeNet HSM Client

NOTE: These branding changes apply to the documentation only. The SafeNet HSM software and utilities continue to use the old names.

Document Conventions

This section provides information on the conventions used in this template.

Notes

Notes are used to alert you to important or helpful information. These elements use the following format:

NOTE: Take note. Contains important or helpful information.

Cautions

Cautions are used to alert you to important information that may help prevent unexpected results or data loss. These elements use the following format:

CAUTION: Exercise caution. Caution alerts contain important information that may help prevent unexpected results or data loss.

Warnings

Warnings are used to alert you to the potential for catastrophic data loss or personal injury. These elements use the following format:

WARNING: Be extremely careful and obey all safety and security measures. In this situation you might do something that could result in catastrophic data loss or personal injury.

Command Syntax and Typeface Conventions

Convention	Description
bold	The bold attribute is used to indicate the following: Command-line commands and options (Type **dir /p**.) Button names (Click **Save As**.) Check box and radio button names (Select the **Print Duplex** check box.) Window titles (On the **Protect Document** window, click **Yes**.)

Convention	Description
	Field names (**User Name:** Enter the name of the user.) Menu names (On the **File** menu, click **Save**.) (Click **Menu** > **Go To** > **Folders**.) User input (In the **Date** box, type **April 1**.)
italic	The italic attribute is used for emphasis or to indicate a related document. (See the *Installation Guide* for more information.)
`Consolas`	Denotes syntax, prompts, and code examples.

Support Contacts

If you encounter a problem while installing, registering or operating this product, please make sure that you have read the documentation. If you cannot resolve the issue, contact your supplier or Gemalto Customer Support. Gemalto Customer Support operates 24 hours a day, 7 days a week. Your level of access to this service is governed by the support plan arrangements made between Gemalto and your organization. Please consult this support plan for further information about your entitlements, including the hours when telephone support is available to you.

Contact Method	Contact Information	
Address	Gemalto 4690 Millennium Drive Belcamp, Maryland 21017, USA	
Phone	US	1-800-545-6608
	International	1-410-931-7520
Technical Support Customer Portal	https://serviceportal.safenet-inc.com Existing customers with a Technical Support Customer Portal account can log in to manage incidents, get the latest software upgrades, and access the Gemalto Knowledge Base.	

1
Introduction

Overview

This document is intended to guide security administrators through the steps for the Oracle Database 11g and 12c Integration with SafeNet Luna HSM, and also covers the necessary information to install, configure and integrate Oracle Database Transparent Data Encryption (TDE) with SafeNet Luna HSM.

TDE provides the infrastructure necessary for implementing encryption. It enables to encrypt sensitive data stored in application table columns (such as credit card numbers etc.) or application tablespaces, the containers for all objects stored in a database TDE prevents data theft of confidential data stored on media. The motivation for the Oracle TDE to use the SafeNet Luna HSM for EKM is because of the following reasons:

- It is used to store the master encryption keys used for transparent data encryption. And the master encryption key is never exposed in insecure memory.
- It also provides more secure computational storage.
- SafeNet Luna HSM is a more secure alternative to the Oracle wallet.

3rd Party Application Details

- Oracle Database 12c R1
- Oracle Database 11g R2
- Oracle Database 11g R1

Supported Platforms

The following platforms are tested with SafeNet Luna HSM:

Oracle Database 12c R1 version: 12.1.0.2

Platforms Tested	SafeNet Luna HSM Client Software Version
Red Hat Enterprise Linux 6.5 64-bit	6.x (6.1.x,6.2.1) 5.x (v5.0.x, 5.1.x, 5.2.x, 5.3.x, 5.4.x)

Platforms Tested	SafeNet Luna HSM Client Software Version
Red Hat Enterprise Linux 7 64-bit	6.x (6.1.x)
AIX 7.1 64 bit Red Hat Enterprise Linux 6.6 64-bit	6.x (6.2.0,6.2.1)

NOTE: On AIX 7.1 64 bit, for SafeNet Luna HSM Client Software Version v6.2.1 integration is tested with Appliance firmware v6.10.9 only.

Oracle Database 12c R1 version: 12.1.0.1

Platforms Tested	Luna Client Software Version
Red Hat Enterprise Linux 6.5 64-bit	5.x (v5.0.x, 5.1.x, 5.2.x, 5.3.x, 5.4.x)

Oracle Database 11g R2 version: 11.2.0.4

Platforms Tested	Luna Client Software Version
SUSE Linux Enterprise Server 10 SP4(x86_64) SUSE Linux Enterprise Server 11 SP3(x86_64) AIX 7.1 64 bit AIX 6.1 64 bit	6.x (6.2.1)
Oracle Linux 5.8 64-bit Red Hat Enterprise Linux 6.5 64-bit	5.x (v5.0.x, 5.1.x, 5.2.x, 5.3.x, 5.4.x)

Oracle Database 11g R2 version: 11.2.0.3

Platforms Tested	Luna Client Software Version
Solaris 10 SPARC 64-bit Solaris 11.1 SPARC 64-bit Red Hat Enterprise Linux 5.2 64-bit Red Hat Enterprise Linux 6.0 64-bit Red Hat Enterprise Linux 6.2 64-bit Red Hat Enterprise Linux 6.3 32-bit HP-UX 11.31 ia64 AIX 6.1 64-bit Aix 7.1 64 bit	5.x (v5.0.x, 5.1.x, 5.2.x, 5.3.x, 5.4.x)

Platforms Tested	Luna Client Software Version
Solaris 10 SPARC 64-bit	4.x (v4.4.x, 4.5.x)

Oracle Database 11g R2 version: 11.2.0.2

Platforms Tested	Luna Client Software Version
Red Hat Enterprise Linux 5 64-bit Solaris 10 SPARC 64-bit AIX 6.1 64-bit HP-UX 11.31 ia64 Oracle Enterprise Linux 5 64-bit	4.x (v4.4.x, 4.5.x)
Oracle Enterprise Linux 5 64-bit HP-UX 11.31 ia64 Solaris 10 SPARC 64-bit Windows Server 2008 R2 Windows Server 2008 R2 Windows Server 2008 32-bit Aix 7.1 x64	5.x (v5.0.x, 5.1.x, 5.2.x, 5.3.x, 5.4.x)

Oracle Database 11g R2 version: 11.2.0.1

Platforms Tested	Luna Client Software Version
Red Hat Enterprise Linux 5 32/64 bit Solaris 10 SPARC 64-bit Solaris 10 x86/64	4.x (v4.4.x, 4.5.x)
Aix 7.1 x64	5.x (v5.0.x, 5.1.x, 5.2.x, 5.3.x, 5.4.x)

Oracle Database 11g R1 version: 11.1.0.7

Platforms Tested	Luna Client Software Version
Windows Server 2003 SP2 32/64 bit Solaris 10 SPARC 64-bit	4.x (v4.4.x, 4.5.x)

Oracle Database 11g R1 version: 11.1.0.6

Platforms Tested	Luna Client Software Version
Windows Server 2003 SP2 32-bit Red Hat Enterprise Linux 5.0 32-bit Solaris 10 SPARC 64-bit	4.x (v4.4.x, 4.5.x)

Multiple Database Support for Single Partition

Oracle Version	Partition Configuration	Key Lifecycle	Encryption wallet creation
11.2.0.3	Requires a unique dedicated partition per Oracle database/cluster	New Install	TDE and TSE keys are created
		Key Migrate	New TDE key is created but the TSE key remains the same
12.1.0.1	Requires a unique dedicated partition per Oracle database/cluster	New Install	TDE and TSE keys are created
		Key Migrate	New TDE key is created but the TSE key remains the same
12.1.0.2	Supports both unique dedicated partitions per Oracle database/cluster and also a single shared partition configuration for all Oracle databases/clusters	New Install	Only TDE key is created
		Key Migrate	New TDE key is created

Known issues

Oracle Database Version	Platform	Oracle Known Issues	Oracle Patch/Workaround
11gR1 (11.1.0.6)	Solaris 10 SPARC 64-bit	ORA-03113: end-of-file on communication channel (On encrypting a column using Luna SA HSM key)	8211698 (p8211698_111060_SOLARIS64.zip)
11gR1 (11.1.0.7)	Solaris 10 SPARC 64-bit	ORA-03113: end-of-file on communication channel (On encrypting a column using Luna SA HSM key	8211698 (p8211698_11107_Solaris-64.zip)

Oracle Database Version	Platform	Oracle Known Issues	Oracle Patch/Workaround
11gR1 (11.1.0.7)	Solaris 10 SPARC 64-bit	Support multiple HSM slots/partitions	9453959 (p9453959_11107_Solaris-64.zip) Note: This patch includes the above mentioned patch (8211698)
11gR1 (11.1.0.7)	Windows Server 2003 64-bit	ORA-28376: cannot find PKCS11 library	Rename the 64 folder to 32 under %SYSTEMDRIVE%\oracle\extapi\64\hsm\safenet\4.4.1\ to %SYSTEMDRIVE%\oracle\extapi\32\hsm\safenet\4.4.1\
11gR1 (11.1.0.7)	Windows Server 2003 32/64-bit	ORA-00603: ORACLE server session terminated by fatal error (On creating a encrypted tablespace using Luna SA HSM key)	None.
11gR2 (11.2.0.1)	Solaris 10 SPARC 64-bit	Support multiple HSM slots/partitions	9229896 (p9229896_112010_SOLARIS64.zip)
	IBM AIX 6.1 on POWER Systems (64-bit)		(p9229896_112010_AIX64-5L.zip)
	RHEL 32-bit		(p9229896_112010_LINUX.zip)
11gR2 (11.2.0.1)	Windows Server 2008/R2 64-bit	ORA-28376: cannot find PKCS11 library	(p10245351_112010_MSWIN-x86-64.zip)
11gR2 (11.2.0.2)	Windows Server 2008 R2 Windows Server 2008 (32-bit)	ORA-28353: failed to open wallet	(p13413154_112020_WINNT.zip)

Prerequisites

SafeNet Network HSM Setup

Refer to the SafeNet Network HSM documentation for installation steps and details regarding the configuration and setup of the box on Windows/Unix systems. Before you get started ensure the following:

- SafeNet Network HSM appliance and a secure admin password.
- SafeNet Network HSM, and a hostname, suitable for your network.
- SafeNet Network HSM network parameters are set to work with your network.
- Initialize the HSM on the SafeNet Network HSM appliance.
- Create and exchange certificates between the SafeNet Network HSM and your Client system.
- Create a partition on the HSM, remember the partition password that will be later used by Microsoft Identity Manager.
- Register the Client with the partition. And run the "vtl verify" command on the client system to display a partition from SafeNet Network HSM. The general form of command is “C:\Program Files\SafeNet\LunaClient> vtl verify” for Windows and “/usr/safenet/lunaclient/bin/vtl verify” for Unix.
- Enabled Partition "Activation" and "Auto Activation" (Partition policy settings 22 and 23 (applies to SafeNet Network HSM with Trusted Path Authentication [which is FIPS 140-2 level 3] only).

SafeNet PCI HSM Setup

Please refer to the **SafeNet PCI HSM** documentation for installation steps and details regarding configuring and setting up the box on Unix systems. Before you get started ensure the following:

- Initialize the HSM on the Luna PCI appliance.
- Create a partition on the HSM.
- Enable Partition "Activation" and "Auto Activation" (Partition policy settings 22 and 23 (applies to SafeNet PCIe HS, with Trusted Path Authentication [which is FIPS 140-2 level 3] only).

SafeNet Luna HSM FIPS Mode

This integration is also tested with SafeNet Luna HSM in FIPS mode.

SafeNet Luna HSM HA (High-Availability) Setup

Please refer to the SafeNet Luna HSM documentation for HA steps and details regarding configuring and setting up two or more HSM boxes on Windows and UNIX systems. You must enabled the HAOnly setting in HA for failover to work so that if primary goes down by some reason all calls automatically routed to secondary till primary gets up again.

Important: If you are using the Luna SA 5.2.1 or above (Firmware 6.10.1 or above) you need the following setting in Chrystoki.conf (UNIX) and Crystoki.ini (Windows) along with HAOnly setting enabled.

UNIX:

```
Misc = {
PE1746Enabled=0;
}
```

Windows:

```
[Misc]
PE1746Enabled=0
```

NOTE: Above setting along with HAOnly enabled required for HA failover to work uninterruptedly on Luna SA 5.2.1 or above (Firmware 6.10.1 or above).

Oracle Database Setup

Oracle Database must be installed on the target machine to carry on with the integration process. For a detailed installation procedure of Oracle Database, refer to the Oracle Database Documentation.

2

Integrating SafeNet Luna HSM with Oracle Database 12c R1

Setting up Luna Client for Transparent Data Encryption

To set up SafeNet Luna HSM for Transparent Data Encryption, perform the following:

Copy the SafeNet Luna HSM PKCS#11 library to the specified directory structure to ensure that the oracle database is able to find this library. Use the following directory structure:

/opt/oracle/extapi/[32,64]/hsm/{Vendor}/{Version}/libXX.ext (Linux/Solaris/AIX/ HPUX)
%SYSTEMDRIVE%\oracle\extapi\[32,64]\hsm\{Vendor}\{Version}\libXX.ext (Windows)

For example:
/opt/oracle/extapi/64/hsm/safenet/5.4.1/libCryptoki2_64.so (RHEL)

Where:

- [32, 64] specifies whether the supplied binary is 32-bits or 64-bits.
- Vendor stands for the name of the vendor supplying the library.
- Version refers to the version of the library. This should preferably be in a format:

  number.number.number. The API name requires no special format. However, the XX must be prefixed with the word lib, as illustrated in the syntax. The extension, ext needs to be replaced by the extension of the library file.

NOTE: Only one PKCS#11 library is supported at a time.

Oracle user should have the read/write permission of the above directory and file and after logged on as oracle you need to export the following variables:

```
export ORACLE_SID=orcl
export ORACLE_BASE=/u01/app/oracle (oracle installation directory)
export ORACLE_HOME=$ORACLE_BASE/product/12.1.0/dbhome_1
export PATH=$PATH:$ORACLE_HOME/bin
export TNS_ADMIN=$ORACLE_HOME/network/admin
```

Generating a Master Encryption Key for HSM-Based Encryption

To start using HSM-based encryption, you need to have a master encryption key that will be stored inside the HSM. The master encryption key is used to encrypt or decrypt column/tablespace encryption keys inside the HSM. HSM can be used in the following ways to protect the Master Encryption Key:

- An existing Unified Master Encryption Key can be migrated onto the HSM.
- A Unified Master Encryption Key can be directly generated onto the HSM.
- HSM Auto-Login wallet use for TDE.

Migrating Master Encryption Key onto the HSM

In order to migrate a Master Encryption Key for HSM-Based Encryption, perform the following instructions:

NOTE: It is assumed that no software-based wallet is yet created in the directory you would specify to create one.

To test TDE with Luna HSM, perform the following:

Verify that the 'traditional' software-based wallet is working fine:

1. Add the following to your $ORACLE_HOME/network/admin/sqlnet.ora `file:`

   ```
   ENCRYPTION_WALLET_LOCATION = (SOURCE = (METHOD = FILE) (METHOD_DATA = (DIRECTORY = <path to the oracle wallet directory>)))
   ```

2. Start the database:

   ```
   $ sqlplus / as sysdba
   ```

   If the database is not yet started, you can start it using:

   ```
   SQL> startup;
   ```

3. Grant ADMINISTER KEY MANAGEMENT or SYSKM privilege to SYSTEM and any user that you want to use.

   ```
   SQL> GRANT ADMINISTER KEY MANAGEMENT TO SYSTEM;

   SQL> commit;
   ```

4. Connect to the database as ‘system’:

   ```
   SQL> connect system/<password>
   ```

   

   NOTE: Password for ‘system’ can be set during Oracle installation. All dbapasswords throughout this document has been set to “temp123#”.

5. Run the ADMINISTER KEY MANAGEMENT SQL statement to create the keystore.

   ```
   SQL> ADMINISTER KEY MANAGEMENT CREATE KEYSTORE 'keystore_location' IDENTIFIED BY software_keystore_password;
   ```

NOTE: 'keystore_location' is the path to oracle wallet directory that you set in the sqlnet.ora file and 'software_keystore_password' must have length more than or equal to 8 characters.

6. Run the ADMINISTER KEY MANAGEMENT SQL statement to open the keystore.

```
SQL> ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN IDENTIFIED BY software_keystore_password;
```

7. Set the master encryption key in the software keystore.

```
SQL> ADMINISTER KEY MANAGEMENT SET KEY IDENTIFIED BY software_keystore_password WITH BACKUP
USING 'backup_identifier';
```

NOTE: WITH BACKUP creates a backup of the keystore. You must use this option for password based keystores. Optionally, you can use the USING clause to add a brief description of the backup. Enclose this description in single quotation marks (' '). This identifier is appended to the named keystore file (for example, ewallet_time_stamp_emp_key_backup.p12, with emp_key_backup being the backup identifier).

8. Create a CUSTOMERS table in the database.

```
SQL> CREATE TABLE CUSTOMERS (ID NUMBER(5), NAME VARCHAR(42), CREDIT_LIMIT NUMBER(10));
```

9. Enter some values in the CUSTOMERS table.

```
SQL> INSERT INTO CUSTOMERS VALUES (001, 'George Bailey', 10000);

SQL> INSERT INTO CUSTOMERS VALUES (002, 'Denial Vettory', 20000);

SQL> INSERT INTO CUSTOMERS VALUES (003, 'MS Dhoni', 30000);

SQL> INSERT INTO CUSTOMERS VALUES (004, 'Shahid Afridi', 40000);
```

10. Encrypt the 'CREDIT_LIMIT' column of the 'CUSTOMERS' table:

```
SQL> ALTER TABLE CUSTOMERS MODIFY (CREDIT_LIMIT ENCRYPT);
```

11. With the next command, the values listed in the encrypted column are returned in clear text; Transparent Data Encryption decrypts them automatically:

```
SQL> SELECT CREDIT_LIMIT FROM CUSTOMERS;
```

12. The next command lists encrypted columns in your database:

```
SQL> SELECT * FROM DBA_ENCRYPTED_COLUMNS;
```

13. Finally, this view contains information about the software keystore itself:

```
SQL> SELECT * FROM V$ENCRYPTION_WALLET;
```

14. Create an encrypted tablespace:

```
SQL> CREATE TABLESPACE SECURESPACE DATAFILE '/u01/app/oracle/oradata/orcl/SECURE01.DBF' SIZE
150M ENCRYPTION DEFAULT STORAGE (ENCRYPT);
```

15. Create a table in the tablespace:

```
SQL> CREATE TABLE EMPLOYEE (ID NUMBER(5),NAME VARCHAR(42),SALARY NUMBER(10)) TABLESPACE
SECURESPACE;
```

16. Insert some values in EMPLOYEE table:

```
SQL> INSERT INTO EMPLOYEE VALUES (001,'JOHN SMITH',15000);

SQL> INSERT INTO EMPLOYEE VALUES (002,'SCOTT TIGER',25000);
```

```
SQL> INSERT INTO EMPLOYEE VALUES (003,'DIANA HAYDEN',35000);
```

17. Display the contents of the EMPLOYEE table with the following command:

    ```
    SQL> SELECT * FROM EMPLOYEE;
    ```

18. Close the keystore:

    ```
    SQL> ADMINISTER KEY MANAGEMENT SET KEYSTORE CLOSE IDENTIFIED BY software_keystore_password;
    ```

19. After closing the keystore execute the command to display the contents again:

    ```
    SQL> SELECT * FROM EMPLOYEE;
    ```

    You will get the following error that means you cannot list the contents of EMPLOYEE table, if keystore is closed.

    ERROR at line 1: ORA-28365: wallet is not open

20. Open the keystore:

    ```
    SQL> ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN IDENTIFIED BY software_keystore_password;

    SQL> exit
    ```

Test if the database can reach the HSM device:

1. Change your `$ORACLE_HOME/network/admin/sqlnet.ora – file:`

   ```
   ENCRYPTION_WALLET_LOCATION = (SOURCE = (METHOD = HSM) (METHOD_DATA =(DIRECTORY = <path to the oracle wallet directory>)))
   ```

2. $ sqlplus / as sysdba

   Connect to the database as 'system':

   ```
   SQL> connect system/<password>
   ```

3. Migrate the wallet onto the HSM device:

   ```
   SQL> ADMINISTER KEY MANAGEMENT SET ENCRYPTION KEY IDENTIFIED BY "hsm_partition_pwd" MIGRATE USING software_keystore_password WITH BACKUP USING 'backup_identifier';
   ```

   NOTE: "hsm_partition_pwd" is the password for the HSM partition where the Master Encryption Key would be generated. The migrate using software_keystore_password string re-encrypts the Transparent Data Encryption column keys and tablespace keys with the new HSM based master key. The software_keystore_password is the password given the software wallet in step 1.

4. With the next command, the values listed in the encrypted column are returned in clear text; Transparent Data Encryption decrypts them automatically, now using the HSM master key:

   ```
   SQL> SELECT CREDIT_LIMIT FROM CUSTOMERS;
   ```

5. Change the password of software keystore to same as HSM partition password.

   ```
   SQL> ADMINISTER KEY MANAGEMENT ALTER KEYSTORE PASSWORD IDENTIFIED BY software_keystore_password SET hsm_partition_pwd WITH BACKUP USING 'backup_identifier';
   ```

   From now onwards when you open the keystore, it will open both software-based keystore as well as HSM-based keystore

6. Close the keystore:

   ```
   SQL> ADMINISTER KEY MANAGEMENT SET KEYSTORE CLOSE IDENTIFIED BY "hsm_partition_pwd";
   ```

7. Open the keystore:

```
SQL> ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN IDENTIFIED BY “hsm_partition_pwd”;
```

This opens both the HSM and the software keystore.

8. Check the wallet information with the following command:

```
SQL> SELECT * FROM V$ENCRYPTION_WALLET;
```

9. Change the password back to the initial password for software based wallet (if you want to do) and use the following syntax to create an auto-login keystore for a software keystore:

```
SQL> ADMINISTER KEY MANAGEMENT CREATE AUTO_LOGIN KEYSTORE FROM KEYSTORE 'keystore_location'
IDENTIFIED BY software_keystore_password;
```

To use the auto-login wallet only on local system use LOCAL AUTO_LOGIN instead of AUTO_LOGIN.

10. Verify that an auto-open software keystore has been created in the oracle wallet directory you specified in the sqlnet.ora file: You will find two wallets in this directory: “ewallet.p12” and “cwallet.sso”; the latter is the auto-open wallet. Move or rename the encryption wallet to ensure that oracle uses auto-open wallet.

```
# mv ewallet.p12 ewallet.p24
```

11. Restart the database and connect to the database as system and open the HSM keystore(software wallet will open automatically):

```
SQL> ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN IDENTIFIED BY "hsm_partition_pwd”;
```

Create HSM Auto Wallet when HSM and Auto-Open Software wallet is in use:

1. Change the sqlnet.ora entries as follows:

```
ENCRYPTION_WALLET_LOCATION = (SOURCE = (METHOD = FILE) (METHOD_DATA = (DIRECTORY =
/etc/oracle/wallet)))
```

2. Rename or move the Auto-Open wallet from the location mentioned in the sqlnet.ora file and move or rename the encryption wallet in to the wallet directory.

```
# mv ewallet.p24 ewallet.p12
```

3. Restart the database and connect as a system.
4. Open the software keystore:

```
SQL> ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN IDENTIFIED BY software_keystore_password;
```

5. Add the HSM secret as a client.

```
SQL> ADMINISTER KEY MANAGEMENT ADD SECRET ‘hsm_partition_password’ FOR CLIENT ‘HSM_PASSWORD’
IDENTIFIED BY software_keystore_password WITH BACKUP USING ‘backup_identifier';
```

NOTE: The secret is the hardware security module password and the client is the HSM_PASSWORD. HSM_PASSWORD is an Oracle-defined client name that is used to represent the HSM password as a secret in the software keystore.

6. Close the software keystore:

```
SQL> ADMINISTER KEY MANAGEMENT SET KEYSTORE CLOSE IDENTIFIED BY software_keystore_password;
```

7. Create (or recreate) Auto-Login keystore.

```
SQL> ADMINISTER KEY MANAGEMENT CREATE AUTO_LOGIN KEYSTORE FROM KEYSTORE ‘keystore_location’
IDENTIFIED BY software_keystore_password;
```

8. Update the sqlnet.ora file to use the hardware security module.

```
ENCRYPTION_WALLET_LOCATION = (SOURCE = (METHOD = HSM) (METHOD_DATA = (DIRECTORY =
/etc/oracle/wallet)))
```

9. Restart the database and connect as a system.
10. Check the wallet information with the following command:

```
SQL> SELECT * FROM V$ENCRYPTION_WALLET;
```

Generating Master Encryption Key directly onto the HSM

In order to generate a Master Encryption Key for HSM-Based Encryption, perform the following instructions:

NOTE: It is assumed that no software or HSM based wallet is yet created.

Setting up Oracle to create Master Encryption Key onto HSM:

1. Add the following to your $ORACLE_HOME/network/admin/sqlnet.ora – file:

```
ENCRYPTION_WALLET_LOCATION = (SOURCE = (METHOD = HSM))
```

2. Start the database:

```
$ sqlplus / as sysdba
```

If the database is not yet started, you can start it using:

```
SQL> startup;
```

3. Grant ADMINISTER KEY MANAGEMENT or SYSKM privilege to SYSTEM and any user that you want to use.

```
SQL> GRANT ADMINISTER KEY MANAGEMENT TO SYSTEM;
SQL> commit;
```

4. Connect to the database as ‘system’:

```
SQL> connect system/<password>
```

NOTE: Password for ‘system’ can be set during Oracle installation. All dbapasswords throughout this document has been set to “temp123#”.

5. Run the ADMINISTER KEY MANAGEMENT SQL statement to open the HSM keystore.

```
SQL> ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN IDENTIFIED BY "hsm_partition_password";
```

6. Set the master encryption key in the software keystore.

```
SQL> ADMINISTER KEY MANAGEMENT SET KEY IDENTIFIED BY "hsm_partition_password";
```

You can see the HSM partition contents to verify the generated keys on HSM, below is the snapshot of HSM partition contents:

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

Partition Name: part7
Partition SN: 152042028
Storage (Bytes): Total=102701, Used=1848, Free=100853
Number objects: 5
Object Label: ORACLE.TDE.HSM.MK.0661286A8C71864F2ABF7891D044154D9A
Object Type: Symmetric Key
Object Label: DATA_OBJECT_SUPPORTED_IDEN
Object Type: Data
Object Label:
```

```
ORACLE.SECURITY.KM.ENCRYPTION.303636313238364138433731383634463241424637383931443034343135
34443941
Object Type: Data
Object Label: DATA_OBJECT_SUPPORTED_IDEN
Object Type: Data
Object Label: ORACLE.TSE.HSM.MK.072AC159D9153C4FF0BF3BF931ED9693850203
Object Type: Symmetric Key
------------------------------------------------------------------------------------------------------------------------
```

7. Create a CUSTOMERS table in the database.

```
SQL> CREATE TABLE CUSTOMERS (ID NUMBER(5), NAME VARCHAR(42), CREDIT_LIMIT NUMBER(10));
```

8. Enter some values in the CUSTOMERS table.

```
SQL> INSERT INTO CUSTOMERS VALUES (001, 'George Bailey', 10000);

SQL> INSERT INTO CUSTOMERS VALUES (002, 'Denial Vettory', 20000);

SQL> INSERT INTO CUSTOMERS VALUES (003, 'MS Dhoni', 30000);

SQL> INSERT INTO CUSTOMERS VALUES (004, 'Shahid Afridi', 40000);
```

9. Encrypt the 'CREDIT_LIMIT' column of the 'CUSTOMERS' table:

```
SQL> ALTER TABLE CUSTOMERS MODIFY (CREDIT_LIMIT ENCRYPT);
```

10. With the next command, the values listed in the encrypted column are returned in clear text; Transparent Data Encryption decrypts them automatically:

```
SQL> SELECT CREDIT_LIMIT FROM CUSTOMERS;
```

11. The next command lists encrypted columns in your database:

```
SQL> SELECT * FROM DBA_ENCRYPTED_COLUMNS;
```

12. Finally, this view contains information about the software keystore itself:

```
SQL> SELECT * FROM V$ENCRYPTION_WALLET;
```

13. Create an encrypted tablespace:

```
SQL> CREATE TABLESPACE SECURESPACE DATAFILE '/u01/app/oracle/oradata/orcl/SECURE01.DBF' SIZE
150M ENCRYPTION DEFAULT STORAGE (ENCRYPT);
```

14. Create a table in the tablespace:

```
SQL> CREATE TABLE EMPLOYEE (ID NUMBER(5),NAME VARCHAR(42),SALARY NUMBER(10)) TABLESPACE
SECURESPACE;
```

15. Insert some values in EMPLOYEE table:

```
SQL> INSERT INTO EMPLOYEE VALUES (001,'JOHN SMITH',15000);

SQL> INSERT INTO EMPLOYEE VALUES (002,'SCOTT TIGER',25000);

SQL> INSERT INTO EMPLOYEE VALUES (003,'DIANA HAYDEN',35000);
```

16. Display the contents of the EMPLOYEE table with the following command:

```
SQL> SELECT * FROM EMPLOYEE;
```

17. Close the keystore:

```
SQL> ADMINISTER KEY MANAGEMENT SET KEYSTORE CLOSE IDENTIFIED BY “hsm_partition_password”;
```

18. After closing the keystore execute the command to display the contents again:

```
SQL> SELECT * FROM EMPLOYEE;
```

    You will get the following error that means you cannot list the contents of EMPLOYEE table, if keystore is closed.

```
ERROR at line 1:
ORA-28365: wallet is not open
```

19. Open the keystore:

```
SQL> ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN IDENTIFIED BY “hsm_partition_password”;
SQL> exit
```

Create HSM Auto Wallet

20. Close the hardware security module if it is open.

```
SQL> ADMINISTER KEY MANAGEMENT SET KEYSTORE CLOSE IDENTIFIED BY "hsm_partition_password";
```

21. Change the sqlnet.ora entries as follows:

```
ENCRYPTION_WALLET_LOCATION = (SOURCE = (METHOD = FILE) (METHOD_DATA = (DIRECTORY =
/etc/oracle/wallet)))
```

22. Create the software keystore in the appropriate location (for example, /etc/oracle/wallet).

```
SQL> ADMINISTER KEY MANAGEMENT CREATE KEYSTORE '/etc/oracle/wallet' IDENTIFIED BY
software_keystore_password;
```

23. Open the software keystore:

```
SQL> ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN IDENTIFIED BY software_keystore_password;
```

24. Add the HSM secret as a client.

```
SQL> ADMINISTER KEY MANAGEMENT ADD SECRET 'hsm_partition_password' FOR CLIENT 'HSM_PASSWORD'
IDENTIFIED BY software_keystore_password WITH BACKUP USING ‘backup_identifier';
```

25. Close the software keystore:

```
SQL> ADMINISTER KEY MANAGEMENT SET KEYSTORE CLOSE IDENTIFIED BY software_keystore_password;
```

26. Create Auto-Login keystore.

```
SQL> ADMINISTER KEY MANAGEMENT CREATE AUTO_LOGIN KEYSTORE FROM KEYSTORE '/etc/oracle/wallet'
IDENTIFIED BY software_keystore_password;
```

27. Update the sqlnet.ora file to use the hardware security module.

```
ENCRYPTION_WALLET_LOCATION = (SOURCE = (METHOD = HSM) (METHOD_DATA = (DIRECTORY =
/etc/oracle/wallet)))
```

    At this stage, close the database and open it one more time and the next time when a TDE operation executes, the hardware security module auto-login keystore opens automatically.

28. Restart the database and connect as a system.
29. Check the wallet information with the following command:

```
SQL> SELECT * FROM V$ENCRYPTION_WALLET;
```

Working with Pluggable Databases (PDB)

A new feature for Oracle Database 12c is Multitenant Architecture, Oracle Multitenant delivers a new architecture that allows a multitenant container database to hold many pluggable databases. The multitenant architecture enables an Oracle database to function as a multitenant container database (CDB) that includes zero, one, or many customer-created pluggable databases (PDBs). A PDB is a portable collection of schemas, schema objects, and non-schema objects that appears to an Oracle Net client as a non-CDB. All Oracle databases before Oracle Database 12c were non-CDBs.

About Containers in a CDB

A container is either a PDB or the root container (also called the root). The root is a collection of schemas, schema objects, and non-schema objects to which all PDBs belong. Every CDB has the following containers:

Exactly one root:

The root stores Oracle-supplied metadata and common users. A common user is a database user known in every container. The root container is named CDB$ROOT.

Exactly one seed PDB:

The seed PDB is a system-supplied template that the CDB can use to create new PDBs. The seed PDB is named PDB$SEED. You cannot add or modify objects in PDB$SEED.

Zero or more user-created PDBs:

A PDB is a user-created entity that contains the data and code required for a specific set of features. For example, a PDB can support a specific application, such as a human resources or sales application. No PDBs exist at creation of the CDB. You add PDBs based on your business requirements.

Managing Pluggable Databases

Purpose of PDBs

You can use PDBs to achieve the following goals:

1. Store data specific to a particular application

   For example, a sales application can have its own dedicated PDB, and a human resources application can have its own dedicated PDB.

2. Move data into a different CDB

   A database is "pluggable" because you can package it as a self-contained unit, and then move it into another CDB.

3. Isolate grants within PDBs

   A local or common user with appropriate privileges can grant **EXECUTE** privileges on a package to **PUBLIC** within an individual PDB.

There are several ways to create a PDB but the most preferred one is to use DBCA utility. It is assumed that you have already created PDBs. For demonstrated purpose in this guide we are using the PDB with named "salespdb".

TDE in Pluggable Databases

Below are the steps to use TDE with Pluggable Databses:

1. Edit the tnsnames.ora file to add a new service for the newly created PDB. By default, the tnsnames.ora file is located in the `ORACLE_HOME/network/admin` directory or in the location set by the TNS_ADMIN environment variable. Ensure that you have properly set the TNS_ADMIN environment variable to point to the correct tnsnames.ora file.

   For Example:

   ```
   salespdb =

   (DESCRIPTION =
         (ADDRESS = (PROTOCOL = TCP)(HOST = localhost)(PORT = 1521))
          (CONNECT_DATA =
            (SERVER = DEDICATED)
            (SERVICE_NAME = salespdb.localdomain)
         )
    )
   ```

   Where, **salespdb** is the new Pluggable database name.

2. Restart the Listener Service.

   ```
   # lsnrctl stop

   # lsnrctl start
   ```

3. Start the **sqlplus** session to connect to PDB.

   ```
   $ sqlplus / as sysdba

   SQL> alter pluggable database all open read write;
   ```

   Pluggable database altered.

   ```
   SQL> Connect system/<system_password>@Pluggable Database Service name
   ```

   For Example:

   ```
   SQL> connect system/temp123#@salespdb Connected.
   ```

4. Run the below grant commands to PDB Admin:

   ```
   SQL> GRANT ADMINISTER KEY MANAGEMENT TO salesadm;
   ```

   Grant succeeded.

   ```
   SQL> GRANT CREATE SESSION TO salesadm;
   ```

   Grant succeeded.

   ```
   SQL> GRANT CONNECT TO salesadm;
   ```

   Grant succeeded.

   ```
   SQL> GRANT DBA TO salesadm;
   ```

   Grant succeeded.

   ```
   SQL> GRANT CREATE ANY TABLE TO salesadm;
   ```

   Grant succeeded.

   ```
   SQL> GRANT UNLIMITED TABLESPACE TO salesadm;
   ```

Grant succeeded.

```
SQL> ALTER USER salesadm PROFILE DEFAULT;
```

User altered.

```
SQL> commit;
```

Commit complete.

Where, **salesadm** is the administrative user name created at the time of creating PDB.

5. Try connecting to PDB with PDB username and you should be able to connect it:

```
SQL> Connect pdbuser/<system_password>@Pluggable Database Service name
```

For Example:

```
SQL> connect salesadm/temp123#@salespdb
Connected.
```

Generating TDE Master Encryption Key for PDB

1. Add the following to your $ORACLE_HOME/network/admin/sqlnet.ora – file:

```
ENCRYPTION_WALLET_LOCATION = (SOURCE = (METHOD = HSM))
```

2. Start the **sqlplus** session to connect to PDB.

```
sqlplus / as sysdba
SQL> Connect <pdb_admin>/<pdb_admin_password>@Pluggable Database Service name
```

For Example:

```
SQL> connect salesadm/temp123#@salespdb
Connected
```

3. Run the ADMINISTER KEY MANAGEMENT SQL statement using the following syntax:

```
ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN IDENTIFIED BY “hsm_partition_password";
```

For example:

```
SQL> ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN IDENTIFIED BY "userpin2";
keystore altered.
```

NOTE: Please make sure that keystore for CDB (root container) is opened and Master key for CDB is generated before opening the keystore and generating the Master key for PDB.Also do not configure HSM auto login for CDB until you generate the master key for PDB (all PDB in case multiple PDB are using the TDE). After generating the Master key for all PDBs you can configure the CDB for auto login and it will work for all PDBs as well.

4. Run the following SQL statement to create the PDB Master key:

```
ADMINISTER KEY MANAGEMENT SET KEY IDENTIFIED BY "hsm_partition_password";
```

For example:

```
SQL> ADMINISTER KEY MANAGEMENT SET KEY IDENTIFIED BY "userpin2";
keystore altered.
```

NOTE: This will generate a new master key and from now onwards any encryption/decryption operations performed within this pdb will use this master key.

5. Create a CUSTOMERS table in the PDB.

```
SQL> CREATE TABLE CUSTOMERS (ID NUMBER(5), NAME VARCHAR(42), CREDIT_LIMIT NUMBER(10));
```

6. Enter some values in the CUSTOMERS table.

```
SQL> INSERT INTO CUSTOMERS VALUES (001, 'George Bailey', 10000);
SQL> INSERT INTO CUSTOMERS VALUES (002, 'Denial Vettory', 20000);
```

7. Encrypt the 'CREDIT_LIMIT' column of the 'CUSTOMERS' table:

```
SQL> ALTER TABLE CUSTOMERS MODIFY (CREDIT_LIMIT ENCRYPT);
```

8. With the next command, the values listed in the encrypted column are returned in clear text; Transparent Data Encryption decrypts them automatically:

```
SQL> SELECT CREDIT_LIMIT FROM CUSTOMERS;
```

9. The next command lists encrypted columns in your databases:

```
SQL> SELECT * FROM DBA_ENCRYPTED_COLUMNS;
```

10. Create an encrypted tablespace:

```
SQL> CREATE TABLESPACE SECURESPACE DATAFILE '/u01/app/oracle/oradata/orcl/salespdb/SECURE01.DBF'
SIZE 150M ENCRYPTION DEFAULT STORAGE (ENCRYPT);
```

11. Create a table in the tablespace:

```
SQL> CREATE TABLE EMPLOYEE (ID NUMBER(5),NAME VARCHAR(42),SALARY NUMBER(10)) TABLESPACE
SECURESPACE;
```

12. Insert some values in EMPLOYEE table:

```
SQL> INSERT INTO EMPLOYEE VALUES (001,'JOHN SMITH',15000);
SQL> INSERT INTO EMPLOYEE VALUES (002,'SCOTT TIGER',25000);
SQL> INSERT INTO EMPLOYEE VALUES (003,'DIANA HAYDEN',35000);
```

13. Display the contents of the EMPLOYEE table with the following command:

```
SQL> SELECT * FROM EMPLOYEE;
```

14. Close the keystore:

```
SQL> ADMINISTER KEY MANAGEMENT SET KEYSTORE CLOSE IDENTIFIED BY “hsm_partition_password”;
```

15. After closing the keystore execute the command to display the contents again:

```
SQL> SELECT * FROM EMPLOYEE;
```

You will get the following error that means you cannot list the contents of EMPLOYEE table, if keystore is closed.

```
ERROR at line 1:
ORA-28365: wallet is not open
```

16. Open the keystore:

```
SQL> ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN IDENTIFIED BY “hsm_partition_password”;
```

The above queries are same as we did for root container but it will use the Master Encryption Key of PDB generated on HSM.

3

Integrating SafeNet Luna HSM with Oracle Database 11g R2

Setting up SafeNet Luna HSM for Transparent Data Encryption

To set up Luna SA/Luna PCI for Transparent Data Encryption, perform the following:

Oracle requires the PKSC#11 library provided by HSM vendor. Copy the Luna SA PKCS#11 library to the specified directory structure recommended by Oracle to ensure that the database is able to find this library. Use the following directory structures for UNIX and Windows respectively:

/opt/oracle/extapi/[32,64]/hsm/{VENDOR}/{VERSION}/libapiname.ext (Linux/Solaris/AIX/ HPUX)

%SYSTEM_DRIVE%\oracle\extapi\[32,64]\hsm\{VENDOR}\{VERSION}\libapiname.ext (Windows)

For example:

/opt/oracle/extapi/64/hsm/safenet/5.1.0/libshim.so (Linux)
/opt/oracle/extapi/64/hsm/safenet/5.1.0/libCryptoki2_64.so (Solaris/AIX/HPUX)
C:\oracle\extapi\64\hsm\safenet\5.1.0\cryptoki.dll (Windows)

NOTE: Rename the libCryptoki2_64.sl to libCryptoki2_64.so on HPUX.

Where:

- [32,64] specifies whether the supplied binary is 32-bits or 64-bits
- VENDOR stands for the name of the vendor supplying the library
- VERSION refers to the version of the library. This should preferably be in a format, number.number.number
- apiname requires no special format. However, the apiname must be prefixed with the word lib, as illustrated in the syntax.
- .ext needs to be replaced by the extension of the library file. This extension is .so on Unix.

NOTE: Only one PKCS#11 library is supported at a time. Oracle user should have the read/write permission of the above directory.

If you are using Luna SA v5.x on Red Hat Enterprise Linux or Oracle Linux then you need to do the following changes in the /etc/Chrystoki.conf file:

- Provide the reference of libshim library in the Crystoki2 section.
- Add the Shim2 section that refers the LibCryptoki file in the /usr/lunasa/lib folder.

```
Chrystoki2 = {
LibUNIX64=/opt/oracle/extapi/64/hsm/safenet/5.0.0/libshim.so;
}
Shim2 = {
LibUNIX64=/usr/lib/libCryptoki2_64.so;
}
```

NOTE: If using 32 bit client then LibUNIX64 must be replaced with LibUNIX and LibCryptoki2_64.so would be LibCryptoki2.so

On HPUX we need to perform the following 2 steps:

- Rename the libCryptoki2_64.sl to libCryptoki2_64.so at the following location:
  ```
  /opt/oracle/extapi/64/hsm/safenet/4.4.1/              (For Luna SA 4.4.1)
  /opt/oracle/extapi/64/hsm/safenet/5.0.0/              (For Luna SA 5.0)
  ```
- After this we need to export these values as given below
  ```
  # LD_PRELOAD="libCsup.so.1 libstd_v2.so.1"
  # export LD_PRELOAD
  ```

Oracle user should have the read/write permission of the above directory and file and after logged on as oracle you need to export the following variables to start/connect the database:

```
export ORACLE_SID=orcl
export ORACLE_BASE=/u01/app/oracle                  (oracle installation directory)
export ORACLE_HOME=$ORACLE_BASE/product/11.2.0/dbhome_1
export PATH=$PATH:$ORACLE_HOME/bin
export TNS_ADMIN=$ORACLE_HOME/network/admin
```

Generating a Master Encryption Key for HSM-Based Encryption

To start using HSM-based encryption, you need to have a master encryption key that will be stored inside the HSM. The master encryption key is used to encrypt or decrypt column encryption keys inside the HSM. HSM can be used in the following ways to protect the Master Encryption Key:

- An existing Unified Master Encryption Key can be migrated onto the HSM.
- A Unified Master Encryption Key can be directly generated onto the HSM.
- HSM Auto-Login wallet use for TDE.

Migrating Master Encryption Key onto the HSM

In order to migrate a Master Encryption Key for HSM-Based Encryption, perform the following instructions:

NOTE: It is assumed that no software-based wallet is yet created in the directory you would specify to create one.

To test TDE with Luna SA HSM, perform the following:

Verify that the 'traditional' software-based wallet is working fine:

1. Add the following to your $ORACLE_HOME\network\admin\sqlnet.ora – file:

   ```
   ENCRYPTION_WALLET_LOCATION = (SOURCE = (METHOD = FILE)(METHOD_DATA = (DIRECTORY = <path to the oracle wallet directory>)))
   ```

2. Start the database:

   ```
   $ sqlplus / as sysoper
   ```

   If the database is not yet started, you can start it using:

   ```
   SQL> startup
   ```

3. Connect to the database as 'system':

   ```
   SQL> connect system/<password>
   ```

   NOTE: Password for 'system' can be set during Oracle installation. All dbapasswords throughout this document has been set to "temp123#".

4. Create an encryption wallet; the master key is added into it automatically; the double quotes are mandatory:

   ```
   SQL> alter system set encryption key identified by "wallet_password";
   ```

   NOTE: "wallet_password" must contain alphanumeric characters and have length more than or equal to 8 characters.

5. Encrypt the 'credit_limit' column of the 'CUSTOMERS' table which is owned by the user 'OE':

   ```
   SQL> alter table oe.customers modify (credit_limit encrypt);
   ```

6. With the next command, the values listed in the encrypted column are returned in clear text; Transparent Data Encryption decrypts them automatically:

   ```
   SQL> select credit_limit from oe.customers where rownum <15;
   ```

7. The next command lists encrypted columns in your database:
   ```
   SQL> select * from dba_encrypted_columns;
   ```

8. Finally, this view contains information about the wallet itself:

   ```
   SQL> select * from v$encryption_wallet;
   ```

9. Create an encrypted tablespace:

   ```
   SQL> CREATE TABLESPACE securespace DATAFILE '/u01/app/oracle/oradata/orcl/secure01.dbf' SIZE 10M ENCRYPTION DEFAULT STORAGE (ENCRYPT);
   ```

10. Create a table in the tablespace:

```
SQL> create table employee (id number(5), name varchar(42), salary number(10)) TABLESPACE
securespace;
```

11. Insert some values in employee table:

```
SQL> Insert into employee values (001,'JOHN SMITH',10000);
SQL> Insert into employee values (002,'SCOTT TIGER',20000);
SQL> Insert into employee values (003,'DIANA HAYDEN',50000);
```

12. Display the contents of the EMPLOYEE table with the following command:

```
SQL> select * from employee;
```

13. Close the wallet:
```
SQL> alter system set encryption wallet close identified by "wallet_password";
```

14. After closing the wallet execute the command to display the contents again:

```
SQL> select * from employee;
```

You will get the following error that means you cannot list the contents of EMPLOYEE table, if wallet is closed.

```
ERROR at line 1:
ORA-28365: wallet is not open
```

Test if the database can reach the HSM device

1. Change your $ORACLE_HOME\network\admin\sqlnet.ora – file:

```
ENCRYPTION_WALLET_LOCATION = (SOURCE = (METHOD = HSM) (METHOD_DATA =   (DIRECTORY = <path to
the oracle wallet directory>)))
```

2. Connect to the database as 'system':
3. Migrate the wallet onto the HSM device:

```
SQL> alter system set encryption key identified by "hsm_partition_pwd" migrate using
"wallet_password;
```

NOTE: "hsm_partition_pwd" is the password for the HSM partition where the Master Encryption Key would be generated. The master key in the HSM device will not be used by encrypted tablespaces; these rely on the software wallet created in step 1. The 'migrate using "wallet_password"' string re-encrypts the Transparent Data Encryption column keys with the new HSM based master key. The "wallet_password" is the password given the software wallet in step 1.

4. With the next command, the values listed in the encrypted column are returned in clear text; Transparent Data Encryption decrypts them automatically, now using the HSM master key:
```
SQL> select credit_limit from oe.customers where rownum <15;
```

5. Close the wallet:
```
SQL> alter system set encryption wallet close identified by "wallet_password";
SQL> exit
```

6. Change the software wallet password to HSM partition password.

```
# orapki wallet change_pwd -wallet [wallet_location/ewallet.p12] -oldpwd <software wallet
password > -newpwd <HSM partition password>
```

NOTE: When the password for both wallet and HSM partition will be same then both wallets will open/close by executing a single command.

NOTE: To change the software wallet password and creating the auto login software wallet oracle provides the Oracle Wallet Manager (owm utility) that can be used instead of orapki utility. To use owm refer the following:

Using Oracle Wallet Manager (owm) for changing password and Auto Login

7. Connect to the database as 'system':

   ```
   SQL> alter system set encryption wallet open identified by “hsm_partition_pwd”
   ```

   This opens both the HSM and the software wallet.

8. Check the wallet information with the following command:

   ```
   SQL> Select * from v$encryption_wallet;
   ```

9. Close the wallet:

   ```
   SQL> alter system set encryption wallet close identified by “hsm_partition_pwd”;
   SQL> exit
   ```

10. Change the password (if you wish to) back to the initial password for software based wallet using orapki and create an auto-login software based wallet.

    ```
    # cd <path to the oracle wallet directory>
    # orapki wallet create -wallet . -auto_login
    ```

NOTE: When prompt for a password, enter the wallet_password. To use the auto-login wallet only on local system use “auto_login_local” instead of “auto_login”.

11. Verify that an auto-open software wallet has been created in the oracle wallet directory you specified in the sqlnet.ora file: You will find two wallets in this directory: “ewallet.p12” and “cwallet.sso”; the latter is the auto-open wallet; rename the encryption wallet ewallet.p12 to ewallet.p24 so that the Transparent Data Encryption does not try to open it. By default oracle opens the encryption wallet first and if it is not there then auto wallet will be selected.

12. Connect to the database as system and open the HSM wallet (the software wallet would be open already):

    ```
    SQL> alter system set encryption wallet open identified by “hsm_partition_pwd”;
    ```

Generating Master Encryption Key directly onto the HSM

In order to generate a Master Encryption Key for HSM-Based Encryption, perform the following instructions:

NOTE: It is assumed that no software or HSM based wallet is yet created.

Setting up Oracle to create Master Encryption Key onto HSM

1. Add the following to your $ORACLE_HOME/network/admin/sqlnet.ora – file:

   ```
   ENCRYPTION_WALLET_LOCATION = (SOURCE = (METHOD = HSM))
   ```

2. Start the database:

   ```
   $ sqlplus / as sysoper
   ```

   If the database is not yet started, you can start it using:

   ```
   SQL> startup
   ```

3. Connect to the database as 'system':

   ```
   SQL> connect system/<password>
   ```

4. Create an encryption wallet. The master key would automatically be created onto the HSM.

   ```
   SQL> alter system set encryption key identified by "hsm_partition_pwd";
   ```

   To use multiple slots or partitions, the syntax to generate master key to a particular slot or partition, use the following syntax:

   ```
   SQL> alter system set encryption key identified by "hsm_partition_pwd|<partition_name>";
   ```

   Above command will be used when you have registered multiple SA partitions and want to choose a particular one partition.

5. Encrypt the 'credit_limit' column of the 'CUSTOMERS' table which is owned by the user 'OE':

   ```
   SQL> alter table oe.customers modify (credit_limit encrypt);
   ```

6. With the next command, the values listed in the encrypted column are returned in clear text; Transparent Data Encryption decrypts them automatically:

   ```
   SQL> select credit_limit from oe.customers where rownum <15;
   ```

7. The next command lists encrypted columns in your database:

   ```
   SQL> select * from dba_encrypted_columns;
   ```

8. Finally, this view contains information about the wallet itself:

   ```
   SQL> select * from v$encryption_wallet;
   ```

9. Create an encrypted tablespace:

   ```
   SQL> CREATE TABLESPACE securespace DATAFILE '/u01/app/oracle/oradata/orcl/secure01.dbf' SIZE 10M
   ENCRYPTION DEFAULT STORAGE (ENCRYPT);
   ```

10. Create a table in the tablespace:

    ```
    SQL> create table employee (id number(5), name varchar(42), salary number(10)) TABLESPACE
    securespace;
    ```

11. Insert some values in employee table:

    ```
    SQL> Insert into employee values (001,'JOHN SMITH',10000);
    SQL> Insert into employee values (002,'SCOTT TIGER',20000);
    SQL> Insert into employee values (003,'DIANA HAYDEN',50000);
    ```

12. Display the contents of the EMPLOYEE table with the following command:

    ```
    SQL> select * from employee;
    ```

13. Close the wallet:
```
SQL> alter system set encryption wallet close identified by “hsm_partition_pwd”;
```
14. After closing the wallet execute the command to display the contents again:
```
SQL> select * from employee;
```

You will get the following error that means you cannot list the contents of EMPLOYEE table, if wallet is closed.
```
ERROR at line 1:
ORA-28365: wallet is not open
```

15. Open the wallet:
```
SQL> alter system set encryption wallet open identified by “hsm_partition_pwd”;
```

Setting up Oracle to create Auto-open HSM

1. When TDE is used in ‘HSM only’ mode (never migrated from an Oracle Wallet):

a. The current entry in sqlnet.ora:
```
ENCRYPTION_WALLET_LOCATION = (SOURCE = (METHOD = HSM))
```
Needs to be changed to:
```
ENCRYPTION_WALLET_LOCATION = (SOURCE = (METHOD = HSM) (METHOD_DATA = (DIRECTORY =
/etc/ORACLE/WALLETS/$ORACLE_SID)))
```

b. Create a (local) auto-open and encryption wallet in /etc/ORACLE/WALLETS/$ORACLE_SID:
```
# orapki wallet create -wallet . -auto_login[_local]
```
When prompt for a password provide a paasword of at least 8 characters. It will be your software wallet password. Remember it for future use.

NOTE: If you want to use that auto login wallet on the local system only then use auto_login_local

c. Add the following entry to the empty wallets to enable an ‘auto-open’ HSM:
```
# mkstore -wrl . -createEntry ORACLE.TDE.HSM.AUTOLOGIN <any-non-empty-string>
```

NOTE: <any-non-empty-string> could be any string and it is used only one time e.g. HDJHEUI3256363

d. Oracle opens the encryption wallet first and if not present then auto wallet. Rename the encryption wallet (ewallet.p12) or move it out of the ‘ENCRYPTION_WALLET_LOCATION’ defined in ‘sqlnet.ora’ to a secure location; do not delete the encryption wallet and do not forget the wallet password.

e. Close the connection to the HSM with
```
SQL> alter system set encryption wallet close identified by “Partition_password”;
```
And open it one last time with
```
SQL> alter system set encryption wallet open identified by “Partition_password”;
```

This will insert “Partition_password” into the auto-open wallet; from now on, no password is required to access encrypted data with the TDE master encryption key stored in an HSM.

2. When TDE was never used before (you are using the TDE first time and want to create HSM auto login):

a. Create a new entry in sqlnet.ora:

```
ENCRYPTION_WALLET_LOCATION = (SOURCE = (METHOD = HSM) (METHOD_DATA = (DIRECTORY = /etc/ORACLE/WALLETS/$ORACLE_SID)))
```

b. Create a (local) auto-open and encryption wallet in /etc/ORACLE/WALLETS/$ORACLE_SID:

```
# orapki wallet create -wallet . -auto_login[_local]
```

When prompt for a password provide a paasword of at least 8 characters. It will be your software wallet password. Remember it for future use.

NOTE: If you want to use that auto login wallet on the local system only then use auto_login_local

c. Add the following entry to the empty wallets to enable an ‘auto-open HSM’:

```
# mkstore -wrl . -createEntry ORACLE.TDE.HSM.AUTOLOGIN <any-non-empty-string>
```

NOTE: <any-non-empty-string> could be any string and it is used only one time e.g. HDJHEUI3256363

d. Oracle opens the encryption wallet first and if not present then auto wallet.Rename the encryption wallet (ewallet.p12) or move it out of the ‘ENCRYPTION_WALLET_LOCATION’ defined in ‘sqlnet.ora’ to a secure location; do not delete the encryption wallet and do not forget the wallet password.

e. Create a TDE master encryption key inside the HSM:

```
SQL> alter system set encryption key identified by “Partition_password”;
```

This will insert “Partition_password” into the auto-open wallet; from now on, no password is required to access encrypted data with the TDE master encryption key stored in an HSM.

3. When an HSM and an encryption wallet is in use:

a. Master Key migrated to HSM and sqlnet.ora contains the following entry:

```
ENCRYPTION_WALLET_LOCATION = (SOURCE = (METHOD = HSM) (METHOD_DATA = (DIRECTORY = /etc/ORACLE/WALLETS/$ORACLE_SID)))
```

b. An auto-open HSM requires the wallet to be a (local) auto-open wallet as well; this (local) auto-open wallet will contain the TDE master key from the encryption wallet and the auto-open string for the HSM.

Create an auto-open wallet from the encryption wallet:

```
# orapki wallet create -wallet . -auto_login[_local]
```

c. Continue at 4.c.)

4. When an HSM and a (local) auto-open wallet is in use:

a. After migrating the Master Key to HSM, a (local) auto-open wallet is used and the encryption wallet was either renamed or removed from the "ENCRYPTION_WALLET_LOCATION";
sqlnet.ora contains the following entry:

```
ENCRYPTION_WALLET_LOCATION = (SOURCE = (METHOD = HSM) (METHOD_DATA = (DIRECTORY =
/etc/ORACLE/WALLETS/$ORACLE_SID)))
```

b. Restore the encryption wallet from its secure location or rename it back to the original file name ewallet.p12

c. Add the following entry to the wallets to enable an 'auto-open HSM', applying the wallet password for the encryption wallet:

```
# mkstore -wrl . -createEntry ORACLE.TDE.HSM.AUTOLOGIN <any-non-empty-string>
```

d. Rename the encryption wallet or move it out of the ENCRYPTION_WALLET_LOCATION defined in sqlnet.ora, do not delete the encryption wallet and do not forget the wallet password.

e. Close the connection to the HSM with

```
SQL> alter system set encryption wallet close identified by "Partition_password";
```

And open it one last time with

```
SQL> alter system set encryption wallet open identified by "Partition_password";
```

This will automatically insert "Partition_password" into the auto-open wallet. From now on, no password is required to access encrypted data with the TDE master encryption key stored in an HSM.

5. When only an encryption wallet is in use (no HSM):

a. sqlnet.ora contains this entry:

```
ENCRYPTION_WALLET_LOCATION = (SOURCE = (METHOD = FILE) (METHOD_DATA = (DIRECTORY =
/etc/ORACLE/WALLETS/$ORACLE_SID)))
```

b. An auto-open HSM requires the wallet to be a (local) auto-open wallet as well; this (local) auto-open wallet will contain the TDE master key from the encryption wallet and the auto-open string for the HSM. Create an auto-open wallet from the encryption wallet:

```
# orapki wallet create –wallet . -auto_login[_local]
```

c. Continue at 6.a.)

6. When a (local) auto-open wallet is in use:

a. Add the following entry to the wallets to enable an 'auto-open HSM':

```
# mkstore -wrl . -createEntry ORACLE.TDE.HSM.AUTOLOGIN <any-non-empty-string>
```

b. Change the entry in sqlnet.ora to:

```
ENCRYPTION_WALLET_LOCATION = (SOURCE = (METHOD = HSM) (METHOD_DATA = (DIRECTORY =
/etc/ORACLE/WALLETS/$ORACLE_SID)))
```

c. Migrate the TDE column encryption master key from wallet to HSM with:

```
SQL> alter system set encryption key identified by "Partition_password" migrate using
"wallet_password";
```

This will insert "Partition_password" into the auto-open wallet. From now on, no password is required to access encrypted data with the TDE master encryption key stored in an HSM.

d. Rename the encryption wallet or move it out of the ENCRYPTION_WALLET_LOCATION defined in sqlnet.ora; do not delete the encryption wallet and do not forget the wallet password.

Using Oracle Wallet Manager (owm) for changing password and Auto Login

1. Start Oracle Wallet Manager from Start Menu:
2. Open the software-based wallet and click on 'Change Password'; use the same string you used for the HSM wallet as the new password for the software based wallet in the form "hsm_partition_pwd"; click on "Save", then "Exit".
3. `$ sqlplus / as sysdba`

   Connect to the database as 'system':
   `SQL> alter system set encryption wallet open identified by "hsm_partition_pwd"`

   This opens both the HSM and the software wallet.
4. Close the wallet:

   ```
   SQL> alter system set encryption wallet close identified by "wallet_password";
   SQL> exit
   ```
5. Start Oracle Wallet Manager from Start Menu:
6. Open the software-based wallet, change the password back to the initial password, check 'Auto-Login'; click on 'Save', then 'Exit'
7. Verify that an auto-open software wallet has been created in the oracle wallet directory you specified in the sqlnet.ora file: You will find two wallets in this directory: "ewallet.p12" and "cwallet.sso"; the latter is the auto-open wallet; rename the encryption wallet:
   $ < *path to the oracle wallet directory* >rename ewallet.p12 ewallet.p24
   so that Transparent Data Encryption does not try to open it.
8. Connect to the database as system and open the HSM wallet (the software is already open):

   `SQL> alter system set wallet open identified by "hsm_partition_pwd";`

4

Integrating SafeNet Luna HSM with Oracle Database 11g R1

Setting up SafeNet Luna HSM for Transparent Data Encryption

To set up SafeNet Luna HSM for Transparent Data Encryption, perform the following:

Copy the SafeNet Luna HSM PKCS#11 library to the specified directory structure to ensure that the database is able to find this library. Use the following directory structure:

%SYSTEMDRIVE%\oracle\extapi\[32,64]\hsm\{Vendor}\{Version}\libXX.ext	(Windows)
/opt/oracle/extapi/[32,64]/hsm/{Vendor}/{Version}/libXX.ext	(Solaris)

For example,

C:\oracle\extapi\32\hsm\safenet\4.4.1\cryptoki.dll	(Windows)
/opt/oracle/extapi/32/hsm/safenet/4.4.1/libshim.so	(Solaris)

NOTE: For Windows 64-bit systems, give the 64-bit folder name as 32. This is a known issue with Oracle on Windows 64-bit systems using Luna SA HSM.

Where:

- [32, 64] specifies whether the supplied binary is 32-bits or 64-bits.
- Vendor stands for the name of the vendor supplying the library.
- Version refers to the version of the library. This should preferably be in a format: number.number.number.
- The API name requires no special format. However, the XX must be prefixed with the word lib, as illustrated in the syntax.
- The extension, ext needs to be replaced by the extension of the library file.

NOTE: Only one PKCS#11 library is supported at a time.

Generating a Master Encryption Key for HSM-Based Encryption

To start using HSM-based encryption, you need to have a master encryption key that will be stored inside the HSM. The master encryption key is used to encrypt or decrypt column encryption keys inside the HSM. HSM can be used in the following ways to protect the Master Encryption Key:

- An existing Master Encryption Key for TDE column encryption can be migrated onto the HSM.
- The Master Encryption Keys for TDE column encryption and TDE tablespace encryption can be directly generated onto the HSM.

Migrating Master Encryption Key onto the HSM

In Oracle Database Oracle Database 11g R1, only the master encryption key for TDE Column Encryption can be migrated from the Oracle Wallet to an HSM; the master encryption key for TDE Tablespace Encryption cannot be migrated to an HSM. If an Oracle Database 11g R1 had never seen an Oracle Wallet, both master keys for TDE Column Encryption and TDE Tablespace Encryption can be created in an HSM, but the master key for TDE Tablespace Encryption cannot be re-keyed (rotated) unless the database has been upgraded to Oracle Database 11g Release 2. In order to migrate a Master Encryption Key for HSM-Based Encryption, perform the following instructions:

Apply
http://updates.oracle.com/download/8421211.html
And
http://updates.oracle.com/download/9453959.html

NOTE: It is assumed that no software-based wallet is yet created in the directory you would specify to create one.

To test TDE with Luna SA HSM, perform the following:

Verify that the 'traditional' software-based wallet is working fine

1. Add the following to your $ORACLE_HOME\network\admin\sqlnet.ora – file:
   ```
   ENCRYPTION_WALLET_LOCATION = (SOURCE = (METHOD = FILE)(METHOD_DATA =
   (DIRECTORY = <path to the oracle wallet directory>)))
   ```
2. `C:\sqlplus` (Windows)
   Connect to the database as 'system':

   `$ sqlplus / as sysoper` (Solaris)

   If the database is not yet started, you can start it using:
   ```
   SQL> startup
   ```

   

   NOTE: Password for ‘system’ can be set during Oracle installation.

3. Create an encryption wallet; the master keys for TDE Column Encryption and TDE Tablespace Encryption are added into it automatically; the double quotes are mandatory:

```
SQL> alter system set encryption key identified by “wallet_password”;
```

This creates an Oracle wallet with two master encryption keys: a master encryption key for TDE Column Encryption, and a master encryption key for TDE Tablespace Encryption; the latter cannot be re-keyed (rotated), and not migrated to an HSM unless the database is upgraded to Oracle Database 11g Release 2.

NOTE: “wallet_password” must contain alphanumeric characters and have length more than or equal to 8 characters.

4. Encrypt the 'credit_limit' column of the 'CUSTOMERS' table which is owned by the user 'OE':

```
SQL> alter table oe.customers modify (credit_limit encrypt using 'AES256' no salt 'nomac');
```

5. With the next command, the values listed in the encrypted column are returned in clear text; Transparent Data Encryption decrypts them automatically:

```
SQL> select credit_limit from oe.customers where rownum <15;
```

6. The next command lists encrypted columns in your database:

```
SQL> select * from dba_encrypted_columns;
```

7. Finally, this view contains information about the wallet itself:

```
SQL> select * from v$encryption_wallet;
```

8. Create an encrypted tablespace:

On Windows:

```
SQL>    CREATE TABLESPACE securespace DATAFILE
        'C:\app\Administrator\oradata\orcl\secure01.dbf'   SIZE 10M ENCRYPTION using 'AES256'
        DEFAULT STORAGE (ENCRYPT);
```

On Solaris:

```
SQL>    CREATE TABLESPACE securespace DATAFILE ‘/u01/app/oracle/oradata/orcl/secure01.dbf’
        SIZE 10M ENCRYPTION using 'AES256' DEFAULT STORAGE (ENCRYPT);
```

9. Close the wallet.

```
SQL> alter system set encryption wallet close;
```

```
 SQL> exit
```

Test if the database can reach the HSM device

Change your $ORACLE_HOME\network\admin\sqlnet.ora – file:

```
ENCRYPTION_WALLET_LOCATION = (SOURCE = (METHOD = HSM)(METHOD_DATA =
(DIRECTORY = <path to the oracle wallet directory>)))
```

1. `C:\sqlplus` (Windows)
   Connect to the database as 'system':

```
$ sqlplus / as sysoper          (Solaris)
```

2. Migrate the TDE Column Encryption master key onto the HSM device:

```
SQL> alter system set encryption key identified by “hsm_partition_pwd” migrate using
“wallet_password;
```

"hsm_partition_pwd" is the password for the HSM partition where the Master Encryption Key would be generated. The master key in the HSM device will not be used by encrypted tablespaces; these rely on the software wallet created in step 1. The 'migrate using "wallet_password"' string re-encrypts the TDE Column Encryption table keys with the new HSM based master key. The "wallet_password" is the password given the software wallet in step 1.

3. With the next command, the values listed in the encrypted column are returned in clear text, Transparent Data Encryption decrypts them automatically, now using the HSM master key:

   ```
   SQL> select credit_limit from oe.customers where rownum <15;
   ```

4. Close the wallet:

   ```
   SQL> alter system set encryption wallet close;
   SQL> exit
   ```

5. Start Oracle Wallet Manager from Start Menu:

6. Open the software-based wallet and click on 'Change Password'; use the same string you used for the HSM wallet as the new password for the software based wallet in the form "hsm_partition_pwd"; click on "Save", then "Exit".

7. `C:\sqlplus` (Windows)

   Connect to the database as 'system':

   `$ sqlplus / as sysoper` (Solaris)

   SQL> alter system set encryption wallet open identified by "hsm_partition_pwd"

   This opens both the HSM and the software wallet.

8. Close the wallet:

   ```
   SQL> alter system set encryption wallet close;
   ```

   ```
   SQL> exit
   ```

9. Start Oracle Wallet Manager from Start Menu:

10. Open the software-based wallet, change the password back to the initial password, check 'Auto-Login'; click on 'Save', then 'Exit'

11. Verify that an auto-open software wallet has been created in the oracle wallet directory you specified in the sqlnet.ora file: You will find two wallets in this directory: "ewallet.p12" and "cwallet.sso"; the latter is the auto-open wallet; rename the encryption wallet:
    C:\< *path to the oracle wallet directory* >rename ewallet.p12 ewallet.p24
    so that Transparent Data Encryption does not try to open it.

12. Connect to the database as system and open the HSM wallet (the software is already open):

    ```
    SQL> alter system set encryption wallet open identified by "hsm_partition_pwd";
    ```

Generating Master Encryption Key directly onto the HSM

In order to generate a Master Encryption Key for HSM-Based Encryption, perform the following instructions:

NOTE: It is assumed that no wallet or HSM based master key is yet created.

Setting up Oracle to create Master Encryption Key onto HSM:

1. Add the following to your $ORACLE_HOME\network\admin\sqlnet.ora – file:
   ```
   ENCRYPTION_WALLET_LOCATION = (SOURCE = (METHOD = HSM))
   ```
2. `C:\sqlplus` (Windows)

   Connect to the database as 'system':
   `$ sqlplus / as sysoper` (Solaris)
   If the database is not yet started, you can start it using:
   ```
   SQL> startup
   ```

   NOTE: Password for 'system' can be set during Oracle installation. All dbapasswords throughout this document has been set to "temp123#".

3. The master keys for TDE Column Encryption and TDE Tablespace Encryption would automatically be created onto the HSM.

   ```
   SQL> alter system set encryption key identified by "hsm_partition_pwd";
   ```

4. Encrypt the 'credit_limit' column of the 'CUSTOMERS' table which is owned by the user 'OE':

   ```
   SQL> alter table oe.customers modify (credit_limit encrypt using 'AES256' 'nomac');
   ```

   NOTE: For Solaris SPARC 64 : "ORA-3113: end of file on communication channel" will be thrown. You need to install patch 8211698 (p8211698_11107_Solaris-64.zip) for Oracle 11g R1 (11.1.0.7.0).

5. With the next command, the values listed in the encrypted column are returned in clear text; Transparent Data Encryption decrypts them automatically:

   ```
   SQL> select credit_limit from oe.customers where rownum <15;
   ```

6. The next command lists encrypted columns in your database:

   ```
   SQL> select * from dba_encrypted_columns;
   ```

7. Finally, this view contains information about the wallet itself:

   ```
   SQL> select * from v$encryption_wallet;
   ```

8. Create an encrypted tablespace:

   On Windows:
   ```
   SQL>    CREATE TABLESPACE securespace DATAFILE
           'C:\app\Administrator\oradata\orcl\secure01.dbf'   SIZE 10M ENCRYPTION using 'AES256'
           DEFAULT STORAGE (ENCRYPT);
   ```

   On Solaris:
   ```
   SQL>    CREATE TABLESPACE securespace DATAFILE '/u01/app/oracle/oradata/orcl/secure01.dbf'
           SIZE 10M ENCRYPTION using 'AES256' DEFAULT STORAGE (ENCRYPT);
   ```

   NOTE: You will receive ORA_3113 while creating an encrypted tablespace using master key from HSM. This is a known issue with Oracle.

9. Close the wallet:

```
SQL> alter system set wallet close;
SQL> exit
```

5

Integrating SafeNet Luna HSM with Multiple Oracle Database 11g R2

Setting up SafeNet Luna HSM for Transparent Data Encryption

To set up SafeNet Luna HSM for Transparent Data Encryption for multiple databases, perform the following:

Oracle requires the PKSC#11 library provided by HSM vendor. Copy the SafeNet Luna HSM PKCS#11 library to the specified directory structure recommended by Oracle to ensure that the database is able to find this library. Use the following directory structures for UNIX and Windows respectively:

/opt/oracle/extapi/[32,64]/hsm/{VENDOR}/{VERSION}/libapiname.ext

%SYSTEM_DRIVE%\oracle\extapi\[32,64]\hsm\{VENDOR}\{VERSION}\libapiname.ext

Where:

- [32,64] specifies whether the supplied binary is 32-bits or 64-bits
- VENDOR stands for the name of the vendor supplying the library
- VERSION refers to the version of the library. This should preferably be in a format, number.number.number
- apiname requires no special format. However, the apiname must be prefixed with the word lib, as illustrated in the syntax.
- ext needs to be replaced by the extension of the library file. This extension is .so on Unix.

NOTE: Only one PKCS#11 library is supported at a time. Oracle user should have the read/write permission of the above directory.

For example,

/opt/oracle/extapi/64/hsm/safenet/5.4.0/libCryptoki2_64.so (Linux)

NOTE: Oracle supports one wallet for each database, so you need to register the multiple partitions for multiple databases (one for each database) on a server hosting oracle database.

If you are using SafeNet Luna HSM v5.x on Red Hat Enterprise Linux or Oracle Linux then you need to do the following changes in the /etc/Chrystoki.conf file:

- Provide the reference of libshim library in the Crystoki2 section.
- Add the Shim2 section that refers the LibCryptoki library.

  For Example:

  64 bit Client:

```
Chrystoki2 = {
LibUNIX64=/opt/oracle/extapi/64/hsm/safenet/5.4.0/libshim.so;
}
Shim2 = {
LibUNIX64=/usr/lib/libCryptoki2_64.so;
}
```

  32 bit Client:

```
Chrystoki2 = {
LibUNIX=/opt/oracle/extapi/64/hsm/safenet/5.4.0/libshim.so;
}
Shim2 = {
LibUNIX=/usr/lib/libCryptoki2.so;
}
```

  Logged on as oracle user and export the following variables:

```
export ORACLE_SID=sales
export ORACLE_BASE=/u01/app/oracle                 (oracle installation directory)
export ORACLE_HOME=$ORACLE_BASE/product/11.2.0/dbhome_1
export PATH=$PATH:$ORACLE_HOME/bin
export TNS_ADMIN=$ORACLE_HOME/network/admin
```

  Here:
  ORACLE_SID (Oracle System Identifier) is the name of unique identifier that is set while configuring the database which is uniquely identified the database from other database running on same computer. In this guide we have configured two databases with SID named SALES and ENGG that are running on port 1521 and 1526 respectively.

Generating a Master Encryption Key for HSM-Based Encryption

To start using HSM-based encryption, you need to have a TDE master key that will be stored inside the HSM. This TDE master encryption key is used to encrypt the table key and tablespace encryption key, which in turn is used to encrypt and decrypt data in the table column and tablespace respectively.

HSM can be used in the following ways to protect the Master Encryption Key:

- An existing Unified Master Encryption Key can be migrated onto the HSM.
- A Unified Master Encryption Key can be directly generated onto the HSM.
- HSM Auto-Login wallet use for TDE.

Migrating Master Encryption Key onto the HSM

In order to migrate a Master Encryption Key for HSM-Based Encryption, perform the following instructions:

NOTE: It is assumed that no software-based wallet is created.

To test TDE with Luna SA HSM, perform the following:

Verify that the 'traditional' software-based wallet is working fine

1. Add the following to your $ORACLE_HOME/network/admin/sqlnet.ora – file:

   ```
   ENCRYPTION_WALLET_LOCATION = (SOURCE = (METHOD = FILE)(METHOD_DATA = (DIRECTORY =
   /etc/orcacle/wallet/$ORACLE_SID)))
   ```

   

   NOTE: Make sure that directory location is exist and oracle user has read/write permission to the above directory.

2. Start the database:

   ```
   $ sqlplus / as sysoper
   ```

   If the database is not yet started, you can start it using:

   ```
   SQL> startup
   ```

3. Connect to the database as ‘system’:

   ```
   SQL> connect system/<password>
   ```

   

   NOTE: Password for ‘system’ can be set during Oracle installation. All dbapasswords throughout this document has been set to “temp123#”.

4. Create an encryption wallet; the master key is added into it automatically; the double quotes are mandatory:

   ```
   SQL> alter system set encryption key identified by “wallet_password”;
   ```

NOTE: "wallet_password" must contain alphanumeric characters and have length more than or equal to 8 characters.

5. Encrypt the 'credit_limit' column of the 'CUSTOMERS' table which is owned by the user 'OE':

```
SQL> alter table oe.customers modify (credit_limit encrypt);
```

6. With the next command, the values listed in the encrypted column are returned in clear text; Transparent Data Encryption decrypts them automatically:

```
SQL> select credit_limit from oe.customers where rownum <15;
```

7. The next command lists encrypted columns in your database:

```
SQL> select * from dba_encrypted_columns;
```

8. Finally, this view contains information about the wallet itself:

```
SQL> select * from v$encryption_wallet;
```

9. Create an encrypted tablespace:

```
SQL> CREATE TABLESPACE securespace DATAFILE '/u01/app/oracle/oradata/sales/secure01.dbf' SIZE
10M ENCRYPTION DEFAULT STORAGE (ENCRYPT);
```

10. Create a table in the tablespace:

```
SQL> create table employee (id number(5), name varchar(42), salary number(10)) TABLESPACE
securespace;
```

11. Insert some values in employee table:

```
SQL> Insert into employee values (001,'JOHN SMITH',10000);
SQL> Insert into employee values (002,'SCOTT TIGER',20000);
SQL> Insert into employee values (003,'DIANA HAYDEN',50000);
```

12. Display the contents of the EMPLOYEE table with the following command:

```
SQL> select * from employee;
```

13. Close the wallet:

```
SQL> alter system set encryption wallet close identified by "wallet_password";
```

14. After closing the wallet execute the command to display the contents again:

```
SQL> select * from employee;
```

    You will get the following error that means you cannot list the contents of EMPLOYEE table, if wallet is closed.

```
ERROR at line 1:
ORA-28365: wallet is not open
```

15. Now logged on as oracle user in another console and export the following:

```
export ORACLE_SID=engg
export ORACLE_BASE=/u01/app/oracle                    (oracle installation directory)
export ORACLE_HOME=$ORACLE_BASE/product/11.2.0/dbhome_1
export PATH=$PATH:$ORACLE_HOME/bin
export TNS_ADMIN=$ORACLE_HOME/network/admin
```

    Execute the above steps to another database and verify that software wallet ewallet.p12 file is created for both databases in the following directory:

    /etc/oracle/wallet/sales/

/etc/oracle/wallet/engg/

Test if the database can reach the HSM device

1. Change your $ORACLE_HOME\network\admin\sqlnet.ora – file:
   ```
   ENCRYPTION_WALLET_LOCATION = (SOURCE = (METHOD = HSM) (METHOD_DATA = (DIRECTORY =
   /etc/oracle/wallet/$ORACLE_SID)))
   ```
2. Start the database:
   ```
   $ sqlplus / as sysoper
   ```
3. Connect to the database as 'system':
   ```
   SQL> connect system/<password>
   ```
4. Migrate the wallet onto the HSM device:
   ```
   SQL> alter system set encryption key identified by “hsm_partition_pwd|partition_name” migrate
   using “wallet_password;
   ```
   “hsm_partition_pwd” is the password for the HSM partition and partition_name is the partition label where the Master Encryption Key would be generated. The master key in the HSM device will not be used by encrypted tablespaces; these rely on the software wallet created in step 1. The 'migrate using “wallet_password”' string re-encrypts the Transparent Data Encryption column keys with the new HSM based master key.
5. With the next command, the values listed in the encrypted column are returned in clear text; Transparent Data Encryption decrypts them automatically, now using the HSM master key:
   ```
   SQL> select credit_limit from oe.customers where rownum <15;
   ```
6. Close the wallet:
   ```
   SQL> alter system set encryption wallet close identified by “hsm_partition_pwd|partition_name”;
   SQL> exit
   ```
7. Change the software wallet password to HSM partition password.
   ```
   # orapki wallet change_pwd -wallet [wallet_location/ewallet.p12] -oldpwd <software wallet
   password > -newpwd <HSM partition password>
   ```
   When the password for both wallet and HSM partition will be same then both wallets will open/close by executing a single command.
8. Start the database:
   ```
   $ sqlplus / as sysoper
   ```
9. Connect to the database as 'system':
   ```
   SQL> connect system/<password>
   ```
10. Open the wallet:
    ```
    SQL> alter system set encryption wallet open identified by “hsm_partition_pwd|partition_name”
    ```
    This opens both the HSM and the software wallet.
11. Check the wallet information with the following command:
    ```
    SQL> Select * from v$encryption_wallet;
    ```
12. Close the wallet:
    ```
    SQL> alter system set encryption wallet close identified by “hsm_partition_pwd|partition_name”;
    SQL> exit
    ```

13. Change the password (if you wish to) back to the initial password for software based wallet using orapki and create an auto-login software based wallet.

```
# cd <path to the oracle wallet directory>
# orapki wallet create -wallet . -auto_login
```

    To use the auto-login wallet only on local system use auto_login_local instead of auto_login.

14. Verify that an auto-open software wallet has been created in the oracle wallet directory you specified in the sqlnet.ora file: You will find two wallets in this directory: “ewallet.p12” and “cwallet.sso”; the latter is the auto-open wallet; rename the encryption wallet: $ < *path to the oracle wallet directory* >rename ewallet.p12 ewallet.p24 so that Transparent Data Encryption does not try to open it.
15. Connect to the database as system and open the HSM wallet (the software is already open):

```
SQL> alter system set encryption wallet open identified by “hsm_partition_pwd|partition_name”;
```

16. Now go to the another console in which you are connected with other database instance whose ORACLE_SID=engg and execute the above commands. At step 4 verify that Master key is moved to the HSM partition.

Generating Master Encryption Key directly onto the HSM

In order to generate a Master Encryption Key for HSM-Based Encryption, perform the following instructions:

NOTE: It is assumed that no software or HSM based wallet is yet created.

Setting up Oracle to create Master Encryption Key onto HSM

1. Add the following to your $ORACLE_HOME/network/admin/sqlnet.ora – file:

```
ENCRYPTION_WALLET_LOCATION = (SOURCE = (METHOD = HSM))
```

2. Start the database:

```
$ sqlplus / as sysoper
```

    If the database is not yet started, you can start it using:

```
SQL> startup
```

3. Connect to the database as ‘system’:

```
SQL> connect system/<password>
```

NOTE: Password for ‘system’ can be set during Oracle installation. All dbapasswords throughout this document has been set to “temp123#”.

4. Create an encryption wallet. The master key would automatically be created onto the HSM.

```
SQL> alter system set encryption key identified by “hsm_partition_pwd|partition_name”;
```

NOTE: Verify that Master Encryption key is generated on HSM partition provided in the above command.

5. Encrypt the 'credit_limit' column of the 'CUSTOMERS' table which is owned by the user 'OE':

```
SQL> alter table oe.customers modify (credit_limit encrypt);
```

6. With the next command, the values listed in the encrypted column are returned in clear text; Transparent Data Encryption decrypts them automatically:

```
SQL> select credit_limit from oe.customers where rownum <15;
```

7. The next command lists encrypted columns in your database:

```
SQL> select * from dba_encrypted_columns;
```

8. Finally, this view contains information about the wallet itself:

```
SQL> select * from v$encryption_wallet;
```

9. Create an encrypted tablespace:

```
SQL> CREATE TABLESPACE securespace DATAFILE '/u01/app/oracle/oradata/sales/secure01.dbf' SIZE
10M ENCRYPTION DEFAULT STORAGE (ENCRYPT);
```

10. Create a table in the tablespace:

```
SQL> create table employee (id number(5), name varchar(42), salary number(10)) TABLESPACE
securespace;
```

11. Insert some values in employee table:

```
SQL> Insert into employee values (001,'JOHN SMITH',10000);
SQL> Insert into employee values (002,'SCOTT TIGER',20000);
SQL> Insert into employee values (003,'DIANA HAYDEN',50000);
```

12. Display the contents of the EMPLOYEE table with the following command:

```
SQL> select * from employee;
```

13. Close the wallet:

```
SQL> alter system set encryption wallet close identified by “hsm_partition_pwd|partition_name”;
```

14. After closing the wallet execute the command to display the contents again:

```
SQL> select * from employee;
```

    You will get the following error that means you cannot list the contents of EMPLOYEE table, if wallet is closed.

```
ERROR at line 1:
ORA-28365: wallet is not open
```

15. Open the wallet:

```
SQL> alter system set encryption wallet open identified by “hsm_partition_pwd|partition_name”;
```

16. Now logged on as Oracle user in another terminal and export ORACLE_SID=engg for second database. Execute the above commands and verify that you are able to generate the Master key on another partition and TDE is working as expected.

Oracle TDE Integration with SafeNet Luna HSM is completed and if you want to create the HSM Auto Wallet then you need to follow the steps provide in the Setting up Oracle to create Auto-open HSM.

6

Integrating SafeNet Luna HSM with Oracle Database 12c R1 RAC

Understanding Oracle RAC

Oracle Real Application Clusters (Oracle RAC) enables an Oracle database to run across a cluster of servers, providing fault tolerance, performance, and scalability with no application changes necessary. Oracle RAC provides high availability for applications by removing the single point of failure with a single server.

Oracle Clusterware is installed into a single home directory, which is called the Grid home. Oracle Clusterware enables servers, referred to as hosts or nodes, to operate as if they are one server, commonly referred to as a cluster. Although the servers are standalone servers, each server has additional processes that communicate with other servers. In this way the separate servers appear as if they are one server to applications and end users. Oracle Clusterware provides the infrastructure necessary to run Oracle RAC. The combined processing power of the multiple servers provides greater availability, throughput, and scalability than is available from a single server.

Non-cluster Oracle databases have a one-to-one relationship between the Oracle database and the instance. Oracle RAC environments, however, have a one-to-many relationship between the database and instances. Oracle RAC databases differ architecturally from non-cluster Oracle databases in that each Oracle RAC database instance also has:

- At least one additional thread of redo for each instance
- An instance-specific undo tablespace

The combined processing power of the multiple servers can provide greater throughput and Oracle RAC scalability than is available from a single server.

Oracle Database RAC Setup

You should familiarize yourself with Oracle Database RAC. Refer to the Oracle Database 12c R1 RAC documentation for more information to install and pre-installation requirements.

The 3 machines utilized are denoted in the setup as follows:

- RAC1.localdomain
- RAC2.localdomain
- RAC3.localdomain

In this demonstration we have created 3 Oracle Homes each having 1 database and 3 instances for every database so the setup has 3x3 Oracle RAC setup. You can scale the setup as per your requirement.

Supported Platforms

The following platforms are supported for SafeNet Luna HSM:

Operating System	SafeNet Luna Client	SafeNet Luna HSM	Oracle Database Software
Red Hat Enterprise Linux 6.6 (64 bit)	Luna Client v6.2.0	Luna SA v6.2.0 f/w 6.10.9	12.1.0.2

Verifying Oracle RAC Installation

Before proceeding for HSM based wallet management, it is assumed that Oracle RAC is setup properly and running at this point, you can verify the RAC running information by executing the following commands on any RAC instances:

```
# crsctl stat res -t
--------------------------------------------------------------------------------
Name           Target  State        Server                   State details
--------------------------------------------------------------------------------
Local Resources
--------------------------------------------------------------------------------
ora.DATA.dg
               ONLINE  ONLINE       rac1                     STABLE
               ONLINE  ONLINE       rac2                     STABLE
               ONLINE  ONLINE       rac3                     STABLE
ora.LISTENER.lsnr
               ONLINE  ONLINE       rac1                     STABLE
               ONLINE  ONLINE       rac2                     STABLE
               ONLINE  ONLINE       rac3                     STABLE
ora.asm
               ONLINE  ONLINE       rac1                     Started,STABLE
               ONLINE  ONLINE       rac2                     Started,STABLE
               ONLINE  ONLINE       rac3                     Started,STABLE
ora.net1.network
               ONLINE  ONLINE       rac1                     STABLE
               ONLINE  ONLINE       rac2                     STABLE
               ONLINE  ONLINE       rac3                     STABLE
ora.ons
               ONLINE  ONLINE       rac1                     STABLE
               ONLINE  ONLINE       rac2                     STABLE
               ONLINE  ONLINE       rac3                     STABLE
```

```
--------------------------------------------------------------------------------
Cluster Resources
--------------------------------------------------------------------------------
ora.LISTENER_SCAN1.lsnr
      1        ONLINE  ONLINE       rac2                     STABLE
ora.LISTENER_SCAN2.lsnr
      1        ONLINE  ONLINE       rac1                     STABLE
ora.LISTENER_SCAN3.lsnr
      1        ONLINE  ONLINE       rac3                     STABLE
ora.MGMTLSNR
      1        ONLINE  ONLINE       rac2                     169.254.110.224 192.
                                                             168.1.102,STABLE
ora.cvu
      1        ONLINE  ONLINE       rac3                     STABLE
ora.mgmtdb
      1        ONLINE  ONLINE       rac2                     Open,STABLE
ora.oc4j
      1        ONLINE  ONLINE       rac3                     STABLE
ora.orcl1rac.db
      1        ONLINE  ONLINE       rac1                     Open,STABLE
      2        ONLINE  ONLINE       rac2                     Open,STABLE
      3        ONLINE  ONLINE       rac3                     Open,STABLE
ora.orcl2rac.db
      1        ONLINE  ONLINE       rac1                     Open,STABLE
      2        ONLINE  ONLINE       rac2                     Open,STABLE
      3        ONLINE  ONLINE       rac3                     Open,STABLE
ora.orcl3rac.db
      1        ONLINE  ONLINE       rac1                     Open,STABLE
      2        ONLINE  ONLINE       rac2                     Open,STABLE
      3        ONLINE  ONLINE       rac3                     Open,STABLE
ora.rac1.vip
      1        ONLINE  ONLINE       rac1                     STABLE
ora.rac2.vip
      1        ONLINE  ONLINE       rac2                     STABLE
ora.rac3.vip
      1        ONLINE  ONLINE       rac3                     STABLE
ora.scan1.vip
```

```
      1        ONLINE  ONLINE       rac2                     STABLE
ora.scan2.vip
      1        ONLINE  ONLINE       rac1                     STABLE
ora.scan3.vip
      1        ONLINE  ONLINE       rac3                     STABLE
--------------------------------------------------------------------------------
```

The setup has 3 databases ORCL1RAC, ORCL2RAC and ORCL3RAC and each database has 3 instances which are running simultaneously on 3 nodes. Below are the details of setup:

DATABASE ORCL1RAC

```
[oracle@rac1 ~]$ srvctl config database -d ORCL1RAC
Database unique name: orcl1rac
Database name: orcl1rac
Oracle home: /u01/app/oracle/product/12.1.0.2/db_1
Oracle user: oracle
Spfile: +DATA/ORCL1RAC/PARAMETERFILE/spfile.301.910221979
Password file: +DATA/ORCL1RAC/PASSWORD/pwdorcl1rac.276.910221645
Domain:
Start options: open
Stop options: immediate
Database role: PRIMARY
Management policy: AUTOMATIC
Server pools:
Disk Groups: DATA
Mount point paths:
Services:
Type: RAC
Start concurrency:
Stop concurrency:
OSDBA group: dba
OSOPER group: oper
Database instances: orcl1rac1,orcl1rac2,orcl1rac3
Configured nodes: rac1,rac2,rac3
Database is administrator managed

[oracle@rac1 ~]$ srvctl status database -d ORCL1RAC
Instance orcl1rac1 is running on node rac1
Instance orcl1rac2 is running on node rac2
```

```
Instance orcl1rac3 is running on node rac3
```

DATABASE ORCL2RAC

```
[oracle@rac2 ~]$ srvctl config database -d ORCL2RAC
Database unique name: orcl2rac
Database name: orcl2rac
Oracle home: /u01/app/oracle/product/12.1.0.2/db_2
Oracle user: oracle
Spfile: +DATA/ORCL2RAC/PARAMETERFILE/spfile.331.910223671
Password file: +DATA/ORCL2RAC/PASSWORD/pwdorcl2rac.306.910223319
Domain:
Start options: open
Stop options: immediate
Database role: PRIMARY
Management policy: AUTOMATIC
Server pools:
Disk Groups: DATA
Mount point paths:
Services:
Type: RAC
Start concurrency:
Stop concurrency:
OSDBA group: dba
OSOPER group: oper
Database instances: orcl2rac1,orcl2rac2,orcl2rac3
Configured nodes: rac1,rac2,rac3
Database is administrator managed
[oracle@rac2 ~]$ srvctl status database -d ORCL2RAC
Instance orcl2rac1 is running on node rac1
Instance orcl2rac2 is running on node rac2
Instance orcl2rac3 is running on node rac3
```

DATABASE ORCL3RAC

```
[oracle@rac3 ~]$ srvctl config database -d ORCL3RAC
Database unique name: orcl3rac
Database name: orcl3rac
```

```
Oracle home: /u01/app/oracle/product/12.1.0.2/db_3
Oracle user: oracle
Spfile: +DATA/ORCL3RAC/PARAMETERFILE/spfile.361.910224445
Password file: +DATA/ORCL3RAC/PASSWORD/pwdorcl3rac.336.910224083
Domain:
Start options: open
Stop options: immediate
Database role: PRIMARY
Management policy: AUTOMATIC
Server pools:
Disk Groups: DATA
Mount point paths:
Services:
Type: RAC
Start concurrency:
Stop concurrency:
OSDBA group: dba
OSOPER group: oper
Database instances: orcl3rac1,orcl3rac2,orcl3rac3
Configured nodes: rac1,rac2,rac3
Database is administrator managed

[oracle@rac3 ~]$ srvctl status database -d ORCL3RAC
Instance orcl3rac1 is running on node rac1
Instance orcl3rac2 is running on node rac2
Instance orcl3rac3 is running on node rac3
```

The V$ACTIVE_INSTANCES view can also display the current status of the instances.

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

$ sqlplus / as sysdba

SQL*Plus: Release 12.1.0.2.0 Production on Wed Apr 27 20:31:35 2016

Copyright (c) 1982, 2014, Oracle.  All rights reserved.


Connected to:
Oracle Database 12c Enterprise Edition Release 12.1.0.2.0 - 64bit Production
```

```
With the Partitioning, Real Application Clusters, Automatic Storage Management, OLAP,
Advanced Analytics and Real Application Testing options

SQL> SELECT inst_name FROM v$active_instances;

INST_NAME
---------------------------------------------------------
rac1.localdomain:orcl1rac1
rac2.localdomain:orcl1rac2
rac3.localdomain:orcl1rac3
---------------------------------------------------------
```

The above details are showing that Oracle RAC 3x3 node setup is working as expected. Now setup SafeNet Luna HSM for Oracle TDE.

Setting up SafeNet Luna HSM for Transparent Data Encryption with Oracle RAC

To set up SafeNet Luna HSM for Transparent Data Encryption for RAC databases, perform the following:

Generating a Master Encryption Key for HSM-Based Encryption

To start using HSM-based encryption, you need to have a master encryption key that will be stored inside the HSM. The master encryption key is used to encrypt or decrypt column encryption keys inside the HSM. HSM can be used in the following ways to protect the Master Encryption Key:

- An existing Unified Master Encryption Key can be migrated onto the HSM.
- A Unified Master Encryption Key can be directly generated onto the HSM.
- Oracle Pluggable database Master Encryption Key generated onto the HSM.

NOTE: Setup has 3 Oracle Homes which will be used for above described scenarios. Each Oracle Home will be used for each scenario

Configuring the PKCS11 Provider on Oracle RAC Instances

To set up SafeNet Luna HSM for TDE with Oracle RAC, kindly perform the following steps on RAC1, RAC2 and RAC3:

Oracle requires the PKSC#11 library provided by HSM vendor. Copy the SafeNet Luna SA PKCS#11 library to the specified directory structure recommended by Oracle to ensure that the database is able to find this library. Use the following directory structures for UNIX and Windows respectively:

/opt/oracle/extapi/[32,64]/hsm/{VENDOR}/{VERSION}/libapiname.ext

%SYSTEM_DRIVE%\oracle\extapi\[32,64]\hsm\{VENDOR}\{VERSION}\libapiname.ext

Where:

- [32,64] specifies whether the supplied binary is 32-bits or 64-bits
- VENDOR stands for the name of the vendor supplying the library
- VERSION refers to the version of the library. This should preferably be in a format, number.number.number
- apiname requires no special format. However, the apiname must be prefixed with the word lib, as illustrated in the syntax.
- .ext needs to be replaced by the extension of the library file. This extension is .so on UNIX.

Only one PKCS#11 library is supported at a time. Oracle user should have the read/write permission of the above directory.

For example,

/opt/oracle/extapi/64/hsm/safenet/6.2.0/libCryptoki2_64.so (Linux)

Below are the commands to create the directory and setup the SafeNet library.

```
# mkdir -p /opt/oracle/extapi/64/hsm/safenet/6.2.0
# cp /usr/safenet/lunaclient/lib/libCryptoki2_64.so /opt/oracle/extapi/64/hsm/safenet/6.2.0/
# chown -R oracle:oinstall /opt/oracle/
# chmod -R 775 /opt/oracle/
```

Migrating Master Encryption Key from software wallet to HSM

In order to migrate a Master Encryption Key for HSM-Based Encryption, perform the following instructions:

1. Create the directory for every database and permit the oracle user to access this directory on RAC1, RAC2 and RAC3 instances:

```
# mkdir -pv /etc/oracle/wallet/ORCL1RAC
# mkdir -pv /etc/oracle/wallet/ORCL2RAC
# mkdir -pv /etc/oracle/wallet/ORCL3RAC
# cd /etc
# chown -R oracle:oinstall oracle/wallet/
# chmod -R 700 oracle/wallet/
```

NOTE: Create the identical directory for storing wallet on all RAC instances or you can use the shared disk for storing the wallet. It will ease our work of copying the wallet manually on all instances. You can use the ASMCA utility to create the ACFS (ASM Cluster File Systems) file and mount this file on disk that will be used by all instances. You can follow the oracle documentation for creating the ACFS file for storing wallet on clustered system.

NOTE: It is assumed that no software-based wallet is yet created in the directory you would specify to create one.

To test TDE with Luna HSM, perform the following:

Verify that the 'traditional' software-based wallet is working fine

2. Add the following to your $ORACLE_HOME/network/admin/sqlnet.ora – file:

   ```
   ENCRYPTION_WALLET_LOCATION = (SOURCE = (METHOD = FILE) (METHOD_DATA = (DIRECTORY = <path to the oracle wallet directory>)))
   ```

   For example:

   Create or add the following to your $ORACLE_HOME/network/admin/sqlnet.ora – file on all instances of RAC (for e.g. RAC1, RAC2 and RAC3):

   ```
   ENCRYPTION_WALLET_LOCATION = (SOURCE = (METHOD = FILE) (METHOD_DATA = (DIRECTORY = /etc/oracle/wallet/ORCL1RAC)))
   ```

3. Start the database:

   ```
   $ sqlplus / as sysdba
   ```

   Restart the database, If the database is not yet started, you can start it using:

   ```
   SQL> startup;
   ```

4. Grant ADMINISTER KEY MANAGEMENT or SYSKM privilege to SYSTEM and any user that you want to use.

   ```
   SQL> GRANT ADMINISTER KEY MANAGEMENT TO SYSTEM;
   ```

   ```
   SQL> commit;
   ```

5. Connect to the database as 'system':

   ```
   SQL> connect system/<password>
   ```

   

   NOTE: Password for 'system' can be set during Oracle installation. All dbapasswords throughout this document has been set to "Temp1234".

6. Run the ADMINISTER KEY MANAGEMENT SQL statement to create the keystore.

   ```
   SQL> ADMINISTER KEY MANAGEMENT CREATE KEYSTORE 'keystore_location' IDENTIFIED BY software_keystore_password;
   ```

   'keystore_location' is the path to oracle wallet directory that was set in the sqlnet.ora file and 'software_keystore_password' must have length more than or equal to 8 characters.

7. Run the ADMINISTER KEY MANAGEMENT SQL statement to open the keystore.

   ```
   SQL> ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN IDENTIFIED BY software_keystore_password;
   ```

   Run the above commands (2 to 7) on all instances RAC1, RAC2, and RAC3.

8. Set the master encryption key in the software keystore.

   ```
   SQL> ADMINISTER KEY MANAGEMENT SET KEY IDENTIFIED BY software_keystore_password WITH BACKUP USING 'backup_identifier';
   ```

   WITH BACKUP creates a backup of the keystore. You must use this option for password-based keystores. Optionally, you can use the USING clause to add a brief description of the backup. Enclose this description in single quotation marks (' '). This identifier is appended to the named keystore file (for example, ewallet_time_stamp_emp_key_backup.p12, with emp_key_backup being the backup identifier).

9. Copy the ewallet.p12 file and backup file created in the directory /etc/oracle/wallet/ORCL1RAC from RAC1 to RAC2 and RAC3 in the same directory as on RAC1.

   ```
   $ scp /etc/oracle/wallet/ORCL1RAC/ewallet* oracle@rac2.localdomain:/etc/oracle/wallet/ORCL1RAC/
   ```

```
$ scp /etc/oracle/wallet/ORCL1RAC/ewallet* oracle@rac3.localdomain:/etc/oracle/wallet/ORCL1RAC/
```

10. Create a CUSTOMERS table in the database.

```
SQL> CREATE TABLE CUSTOMERS (ID NUMBER(5), NAME VARCHAR(42), CREDIT_LIMIT NUMBER(10));
```

11. Enter some values in the CUSTOMERS table.

```
SQL> INSERT INTO CUSTOMERS VALUES (001, 'George Bailey', 10000);

SQL> INSERT INTO CUSTOMERS VALUES (002, 'Denial Vettory', 20000);

SQL> INSERT INTO CUSTOMERS VALUES (003, 'MS Dhoni', 30000);

SQL> INSERT INTO CUSTOMERS VALUES (004, 'Shahid Afridi', 40000);
```

12. Encrypt the 'CREDIT_LIMIT' column of the 'CUSTOMERS' table:

```
SQL> ALTER TABLE CUSTOMERS MODIFY (CREDIT_LIMIT ENCRYPT);
```

13. With the next command, the values listed in the encrypted column are returned in clear text; Transparent Data Encryption decrypts them automatically:

```
SQL> SELECT CREDIT_LIMIT FROM CUSTOMERS;
```

14. The next command lists encrypted columns in your database:

```
SQL> SELECT * FROM DBA_ENCRYPTED_COLUMNS;
```

15. Finally, this view contains information about the software keystore itself:

```
SQL> SELECT * FROM GV$ENCRYPTION_WALLET;
```

16. Create an encrypted tablespace:

```
SQL> CREATE TABLESPACE SECURESPACE DATAFILE '+DATA' SIZE 150M ENCRYPTION DEFAULT STORAGE
(ENCRYPT);
```

17. Create a table in the tablespace:

```
SQL> CREATE TABLE EMPLOYEE (ID NUMBER(5),NAME VARCHAR(42),SALARY NUMBER(10)) TABLESPACE
SECURESPACE;
```

18. Insert some values in EMPLOYEE table:

```
SQL> INSERT INTO EMPLOYEE VALUES (001,'JOHN SMITH',15000);

SQL> INSERT INTO EMPLOYEE VALUES (002,'SCOTT TIGER',25000);

SQL> INSERT INTO EMPLOYEE VALUES (003,'DIANA HAYDEN',35000);
```

19. Display the contents of the EMPLOYEE table with the following command:

```
SQL> SELECT * FROM EMPLOYEE;
```

20. Close the keystore:

```
SQL> ADMINISTER KEY MANAGEMENT SET KEYSTORE CLOSE IDENTIFIED BY software_keystore_password;
```

21. After closing the keystore execute the command to display the contents again:

```
SQL> SELECT * FROM EMPLOYEE;
```

    You will get the following error that means you cannot list the contents of EMPLOYEE table, if keystore is closed.

```
ERROR at line 1:

ORA-28365: wallet is not open
```

Now connect to RAC 2 and RAC3 machine and execute the commands below after opening the wallet on machine.

22. Open the keystore:

```
SQL> ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN IDENTIFIED BY software_keystore_password;
```

23. Now try to access the contents of EMPLOYEE table

```
SQL> SELECT * FROM EMPLOYEE;
```

```
SQL> exit
```

Now onwards when you start the database you need to open the wallet to view the encrypted data. You can set the wallet to Auto-Login that will automatically open when database starts

NOTE: Above commands works on all 3 nodes RAC1, RAC2 and RAC3 after copying the wallet on both instances.

Test if the database can reach the HSM device

24. Change your $ORACLE_HOME/network/admin/sqlnet.ora – file:

```
ENCRYPTION_WALLET_LOCATION = (SOURCE = (METHOD = HSM) (METHOD_DATA = (DIRECTORY = <path to the oracle wallet directory>)))
```

For example:

Change the FILE to HSM in your $ORACLE_HOME/network/admin/sqlnet.ora – file on all instances of RAC (for e.g. RAC1, RAC2 and RAC3):

```
ENCRYPTION_WALLET_LOCATION = (SOURCE = (METHOD = HSM) (METHOD_DATA = (DIRECTORY = /etc/oracle/wallet/ORCL1RAC)))
```

25. $ sqlplus / as sysdba

Connect to the database as 'system':

```
SQL> connect system/<password>
```

26. Migrate the wallet onto the HSM device:

```
SQL> ADMINISTER KEY MANAGEMENT SET ENCRYPTION KEY IDENTIFIED BY "hsm_partition_pwd" MIGRATE USING software_keystore_password WITH BACKUP USING 'backup_identifier';
```

“hsm_partition_pwd” is the password for the HSM partition where the Master Encryption Key would be generated. The migrate using software_keystore_password string re-encrypts the Transparent Data Encryption column keys and tablespace keys with the new HSM based master key. The software_keystore_password is the password of software wallet.

27. Copy the ewallet.p12 and backup files created in the directory /etc/oracle/wallet/ORCL1RAC from RAC1 to RAC2 and RAC3 in the same directory as RAC1.

```
$ scp /etc/oracle/wallet/ORCL1RAC/ewallet* oracle@rac2.localdomain:/etc/oracle/wallet/ORCL1RAC/
```

```
$ scp /etc/oracle/wallet/ORCL1RAC/ewallet* oracle@rac3.localdomain:/etc/oracle/wallet/ORCL1RAC/
```

28. With the next command, the values listed in the encrypted column are returned in clear text; Transparent Data Encryption decrypts them automatically, now using the HSM master key:

```
SQL> SELECT * FROM EMPLOYEE;
```

29. Change the password of software keystore to same as HSM partition password.

```
SQL> ADMINISTER KEY MANAGEMENT ALTER KEYSTORE PASSWORD IDENTIFIED BY software_keystore_password
SET hsm_partition_pwd WITH BACKUP USING ‘backup_identifier';
```

30. Copy the ewallet.p12 and backup files created in the directory /etc/oracle/wallet/ORCL1RAC from RAC1 to RAC2 and RAC3 in the same directory as RAC1.

    From now onwards when you open the keystore, it will open both software-based keystore as well as HSM-based keystore

31. Close the keystore:

```
SQL> ADMINISTER KEY MANAGEMENT SET KEYSTORE CLOSE IDENTIFIED BY “hsm_partition_pwd”;
```

32. Open the keystore:

```
SQL> ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN IDENTIFIED BY “hsm_partition_pwd”;
```

    This opens both the HSM and the software keystore.

33. Check the wallet information with the following command:

```
SQL> SELECT * FROM GV$ENCRYPTION_WALLET;
```

NOTE: Above commands works on all 3 nodes RAC1, RAC2 and RAC3 after copying the wallet on both instances. Copy the wallet from one RAC instance to others after changing the wallet password on an instance.

34. Create the auto-login wallet on all the instances. Change the password back to the initial password for software based wallet (if you want to do) and use the following syntax to create an auto-login keystore for a software keystore:

```
SQL> ADMINISTER KEY MANAGEMENT CREATE AUTO_LOGIN KEYSTORE FROM KEYSTORE 'keystore_location'
IDENTIFIED BY software_keystore_password;
```

    To use the auto-login wallet only on local system use LOCAL AUTO_LOGIN instead of AUTO_LOGIN.

35. Verify that an auto-open software keystore has been created in the oracle wallet directory you specified in the sqlnet.ora file: You will find two wallets in this directory: “ewallet.p12” and “cwallet.sso”; the latter is the auto-open wallet.

NOTE: copy the auto-login wallet and encryption wallet from one RAC instance to others after creating it.

36. Restart the database and connect to the database as system and open the HSM keystore (software wallet will open automatically):

```
SQL> ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN IDENTIFIED BY "hsm_partition_pwd”;
```

Create HSM Auto Wallet when HSM and Auto-Open Software wallet is in use.

At this point when database restarts software wallet get opened automatically but HSM based wallet need to be opened manually. To create the HSM wallet auto open performed the following steps:

37. Change the sqlnet.ora entries as follows on all nodes:

```
ENCRYPTION_WALLET_LOCATION = (SOURCE = (METHOD = FILE) (METHOD_DATA = (DIRECTORY =
/etc/oracle/wallet/ORCL1RAC)))
```

38. Rename the cwallet.sso as it is already present at keystore location. Rename it on all nodes.

39. Restart the database and connect as a system and open the software keystore.

```
SQL> ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN IDENTIFIED BY software_keystore_password;
```

40. Add the HSM secret as a client.

```
SQL> ADMINISTER KEY MANAGEMENT ADD SECRET ‘hsm_partition_password’ FOR CLIENT ‘HSM_PASSWORD’
IDENTIFIED BY software_keystore_password WITH BACKUP USING ‘backup_identifier';
```

The secret is the hardware security module password and the client is the HSM_PASSWORD. HSM_PASSWORD is an Oracle-defined client name that is used to represent the HSM password as a secret in the software keystore.

41. Close the software keystore:

```
SQL> ADMINISTER KEY MANAGEMENT SET KEYSTORE CLOSE IDENTIFIED BY software_keystore_password;
```

42. Create (or recreate) Auto-Login keystore.

```
SQL> ADMINISTER KEY MANAGEMENT CREATE AUTO_LOGIN KEYSTORE FROM KEYSTORE
'/etc/oracle/wallet/ORCL1RAC' IDENTIFIED BY software_keystore_password;
```

NOTE: copy the auto-login wallet and encryption wallet from one RAC instance to others after creating it.

43. Update the sqlnet.ora file to use the hardware security module on all nodes.

```
SQL> ENCRYPTION_WALLET_LOCATION = (SOURCE = (METHOD = HSM) (METHOD_DATA = (DIRECTORY =
/etc/oracle/wallet/ORCL1RAC)))
```

44. Restart the database and connect as a system.
45. Check the wallet information with the following command:

```
SQL> SELECT * FROM GV$ENCRYPTION_WALLET;
```

Generating Master Encryption Key directly onto the HSM

In order to generate a Master Encryption Key for HSM-Based Encryption, perform the following instructions:

NOTE: It is assumed that no software or HSM based wallet is yet created.

1. Add the following to your $ORACLE_HOME/network/admin/sqlnet.ora – file:

```
ENCRYPTION_WALLET_LOCATION = (SOURCE = (METHOD = HSM))
```

For example:

Create or add the following to your $ORACLE_HOME/network/admin/sqlnet.ora file on all instances of RAC (for e.g. RAC1, RAC2, and RAC3):

```
ENCRYPTION_WALLET_LOCATION = (SOURCE = (METHOD = HSM))
```

2. Restart the database and If the database is not yet started, you can start it using:

```
$ srvctl stop database -d ORCL2RAC
```

```
$ srvctl start database -d ORCL2RAC
```

Check the status when database start command completes:

```
$ srvctl status database -d ORCL2RAC
```

```
--------------------------------------------------------------------------------
Instance orcl2rac1 is running on node rac1
Instance orcl2rac2 is running on node rac2
Instance orcl2rac3 is running on node rac3
--------------------------------------------------------------------------------
```

3. Start the database:

```
$ sqlplus / as sysdba
```

4. Grant ADMINISTER KEY MANAGEMENT or SYSKM privilege to SYSTEM and any user that you want to use.

```
SQL> GRANT ADMINISTER KEY MANAGEMENT TO SYSTEM;
SQL> commit;
```

5. Connect to the database as 'system':

```
SQL> connect system/<password>
```

NOTE: Password for 'system' can be set during Oracle installation. All dbapasswords throughout this document has been set to "Temp1234".

6. Run the ADMINISTER KEY MANAGEMENT SQL statement to open the HSM keystore.

```
SQL> ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN IDENTIFIED BY "hsm_partition_password";
```

7. Set the master encryption key in the software keystore.

```
SQL> ADMINISTER KEY MANAGEMENT SET KEY IDENTIFIED BY "hsm_partition_password";
```

You can see the HSM partition contents to verify the generated keys on HSM, below is the snapshot of HSM partition contents:

```
--------------------------------------------------------------------------------------------------
Partition Name:  ORCL2
       Partition SN:    152042028
       Storage (Bytes): Total=102701, Used=1848, Free=100853
       Number objects:  5
       Object Label:  ORACLE.TDE.HSM.MK.0661286A8C71864F2ABF7891D044154D9A
       Object Type:   Symmetric Key
       Object Label:  DATA_OBJECT_SUPPORTED_IDEN
       Object Type:   Data
       Object Label:
ORACLE.SECURITY.KM.ENCRYPTION.3036363132383641384337313836344632414246373839314430343431353443941
       Object Type:   Data
       Object Label:  DATA_OBJECT_SUPPORTED_IDEN
       Object Type:   Data
       Object Label:  ORACLE.TSE.HSM.MK.072AC159D9153C4FF0BF3BF931ED9693850203
```

```
            Object Type:   Symmetric Key

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

8. Create a CUSTOMERS table in the database.

   ```
   SQL> CREATE TABLE CUSTOMERS (ID NUMBER(5), NAME VARCHAR(42), CREDIT_LIMIT NUMBER(10));
   ```

9. Enter some values in the CUSTOMERS table.

   ```
   SQL> INSERT INTO CUSTOMERS VALUES (001, 'George Bailey', 10000);

   SQL> INSERT INTO CUSTOMERS VALUES (002, 'Denial Vettory', 20000);

   SQL> INSERT INTO CUSTOMERS VALUES (003, 'MS Dhoni', 30000);

   SQL> INSERT INTO CUSTOMERS VALUES (004, 'Shahid Afridi', 40000);
   ```

10. Encrypt the 'CREDIT_LIMIT' column of the 'CUSTOMERS' table:

    ```
    SQL> ALTER TABLE CUSTOMERS MODIFY (CREDIT_LIMIT ENCRYPT);
    ```

11. With the next command, the values listed in the encrypted column are returned in clear text; Transparent Data Encryption decrypts them automatically:

    ```
    SQL> SELECT CREDIT_LIMIT FROM CUSTOMERS;
    ```

12. The next command lists encrypted columns in your database:

    ```
    SQL> SELECT * FROM DBA_ENCRYPTED_COLUMNS;
    ```

13. Finally, this view contains information about the software keystore itself:

    ```
    SQL> SELECT * FROM GV$ENCRYPTION_WALLET;
    ```

14. Create an encrypted tablespace:

    ```
    SQL> CREATE TABLESPACE SECURESPACE DATAFILE '+DATA' SIZE 150M ENCRYPTION DEFAULT STORAGE
    (ENCRYPT);
    ```

15. Create a table in the tablespace:

    ```
    SQL> CREATE TABLE EMPLOYEE (ID NUMBER(5),NAME VARCHAR(42),SALARY NUMBER(10)) TABLESPACE
    SECURESPACE;
    ```

16. Insert some values in EMPLOYEE table:

    ```
    SQL> INSERT INTO EMPLOYEE VALUES (001,'JOHN SMITH',15000);

    SQL> INSERT INTO EMPLOYEE VALUES (002,'SCOTT TIGER',25000);

    SQL> INSERT INTO EMPLOYEE VALUES (003,'DIANA HAYDEN',35000);
    ```

17. Display the contents of the EMPLOYEE table with the following command:

    ```
    SQL> SELECT * FROM EMPLOYEE;
    ```

18. Close the keystore:

    ```
    SQL> ADMINISTER KEY MANAGEMENT SET KEYSTORE CLOSE IDENTIFIED BY “hsm_partition_password”;
    ```

19. After closing the keystore execute the command to display the contents again:

    ```
    SQL> SELECT * FROM EMPLOYEE;
    ```

    You will get the following error that means you cannot list the contents of EMPLOYEE table, if keystore is closed.

    ```
    ERROR at line 1:

    ORA-28365: wallet is not open
    ```

20. Now go to RAC2 and RAC3 machine, open the wallet on the node.

```
SQL> ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN IDENTIFIED BY “hsm_partition_password”;
```

21. Now try to access the contents of EMPLOYEE table

```
SQL> select * from employee;
```

Now onwards when you start the database you need to open the HSM based wallet to view the encrypted data. You can set the HSM based wallet to Auto-Login that will automatically open when database starts

Create HSM Auto Wallet

22. Close the hardware security module if it is open.

```
SQL> ADMINISTER KEY MANAGEMENT SET KEYSTORE CLOSE IDENTIFIED BY "hsm_partition_password";
```

23. Change the sqlnet.ora entries as follows on all RAC instances (RAC1, RAC2 and RAC3):

```
ENCRYPTION_WALLET_LOCATION = (SOURCE = (METHOD = FILE) (METHOD_DATA = (DIRECTORY =
/etc/oracle/wallet/ORCL2RAC)))
```

24. Create the software keystore in the appropriate location (for example, /etc/oracle/wallet/ORCL2RAC).

```
SQL> ADMINISTER KEY MANAGEMENT CREATE KEYSTORE '/etc/oracle/wallet/ORCL2RAC' IDENTIFIED BY
software_keystore_password;
```

25. Open the software keystore:

```
SQL> ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN IDENTIFIED BY software_keystore_password;
```

26. Add the HSM secret as a client.

```
SQL> ADMINISTER KEY MANAGEMENT ADD SECRET 'hsm_partition_password' FOR CLIENT 'HSM_PASSWORD'
IDENTIFIED BY software_keystore_password WITH BACKUP USING ‘backup_identifier';
```

27. Close the software keystore:

```
SQL> ADMINISTER KEY MANAGEMENT SET KEYSTORE CLOSE IDENTIFIED BY software_keystore_password;
```

28. Create Auto-Login keystore.

```
SQL> ADMINISTER KEY MANAGEMENT CREATE AUTO_LOGIN KEYSTORE FROM KEYSTORE
'/etc/oracle/wallet/ORCL2RAC' IDENTIFIED BY software_keystore_password;
```

29. Update the sqlnet.ora file to use the hardware security module on all instances.

```
ENCRYPTION_WALLET_LOCATION = (SOURCE = (METHOD = HSM) (METHOD_DATA = (DIRECTORY =
/etc/oracle/wallet/ORCL2RAC)))
```

30. Copy both ewallet.p12 and cwallet.sso files created in the directory /etc/oracle/wallet/ORCL2RAC from RAC1 to RAC2 and RAC3 in the same directory as on RAC1.

```
$ scp /etc/oracle/wallet/ORCL2RAC/* oracle@rac2.localdomain:/etc/oracle/wallet/ORCL2RAC/

$ scp /etc/oracle/wallet/ORCL2RAC/* oracle@rac3.localdomain:/etc/oracle/wallet/ORCL2RAC/
```

At this stage, close the database and open it one more time and the next time when a TDE operation executes, the hardware security module auto-login keystore opens automatically.

31. Restart the database and connect as a system.

32. Check the wallet information with the following command:

```
SQL> SELECT * FROM GV$ENCRYPTION_WALLET;
```

Working with Pluggable Databases (PDB)

A new feature for Oracle Database 12c is Multitenant Architecture, Oracle Multitenant delivers a new architecture that allows a multitenant container database to hold many pluggable databases. The multitenant architecture enables an Oracle database to function as a multitenant container database (CDB) that includes zero, one, or many customer-created pluggable databases (PDBs). A PDB is a portable collection of schemas, schema objects, and non-schema objects that appears to an Oracle Net client as a non-CDB. All Oracle databases before Oracle Database 12c were non-CDBs.

About Containers in a CDB

A container is either a PDB or the root container (also called the root). The root is a collection of schemas, schema objects, and non-schema objects to which all PDBs belong.

Every CDB has the following containers:

- Exactly one root:

  The root stores Oracle-supplied metadata and common users. A common user is a database user known in every container. The root container is named CDB$ROOT.

- Exactly one seed PDB:

  The seed PDB is a system-supplied template that the CDB can use to create new PDBs. The seed PDB is named PDB$SEED. You cannot add or modify objects in PDB$SEED.

- Zero or more user-created PDBs:

  A PDB is a user-created entity that contains the data and code required for a specific set of features. For example, a PDB can support a specific application, such as a human resources or sales application. No PDBs exist at creation of the CDB. You add PDBs based on your business requirements.

Managing Pluggable Databases

Purpose of PDBs

You can use PDBs to achieve the following goals:

- Store data specific to a particular application

  For example, a sales application can have its own dedicated PDB, and a human resources application can have its own dedicated PDB.

- Move data into a different CDB

  A database is "pluggable" because you can package it as a self-contained unit, and then move it into another CDB.

- Isolate grants within PDBs

  A local or common user with appropriate privileges can grant EXECUTE privileges on a package to PUBLIC within an individual PDB.

There are several ways to create a PDB but the most preferred one is to use DBCA utility. It is assumed that you have already created PDBs. For demonstration purpose in this guide we are using the PDB with named “salespdb”.

TDE in Pluggable Databases

Below are the steps to use TDE with Pluggable Databses:

1. Edit the tnsnames.ora file to add a new service for the newly created PDB. By default, the tnsnames.ora file is located in the ORACLE_HOME/network/admin directory or in the location set by the TNS_ADMIN environment variable. Ensure that you have properly set the TNS_ADMIN environment variable to point to the correct tnsnames.ora file.

   For Example:

   ```
   SALESPDB =
     (DESCRIPTION =
        (ADDRESS = (PROTOCOL = TCP)(HOST = scan)(PORT = 1521))
        (CONNECT_DATA =
           (SERVER = DEDICATED)
           (SERVICE_NAME = salespdb)
         )
       )
   ```

   Where, salespdb is the new Pluggable database name. Ensure the above changes on all RAC instances (RAC1, RAC2 and RAC3).

2. Restart the Listener Service.

   ```
   # lsnrctl stop
   # lsnrctl start
   ```

3. Start the sqlplus session to connect to PDB.

   ```
   $ sqlplus / as sysdba
   ```

   SQL> alter pluggable database all open read write;

   Pluggable database altered.

   ```
   SQL> Connect system/<system_password>@Pluggable Database Service name
   ```

   For Example:

   ```
   SQL> connect system/temp123#@salespdb
   Connected.
   ```

4. Run the below grant commands to PDB Admin:

   ```
   SQL> GRANT ADMINISTER KEY MANAGEMENT TO salesadm;
   Grant succeeded.
   SQL> GRANT CREATE SESSION TO salesadm;
   Grant succeeded.
   SQL> GRANT CONNECT TO salesadm;
   Grant succeeded.
   SQL> GRANT DBA TO salesadm;
   Grant succeeded.
   SQL> GRANT CREATE ANY TABLE TO salesadm;
   Grant succeeded.
   SQL> GRANT UNLIMITED TABLESPACE TO salesadm;
   ```

```
Grant succeeded.

SQL> ALTER USER salesadm PROFILE DEFAULT;

User altered.

SQL> commit;

Commit complete.
```

Where, salesadm is the administrative user name created at the time of creating PDB. Run these grant and alter command on all instances and commit.

5. Try connecting to PDB with PDB username and you should be able to connect it:

```
SQL> Connect pdbuser/<system_password>@Pluggable Database Service name
```

For Example:

```
SQL> connect salesadm/temp123#@salespdb

Connected.
```

Generating TDE Master Encryption Key for PDB

1. Add the following to your $ORACLE_HOME/network/admin/sqlnet.ora – file on all instances:

```
ENCRYPTION_WALLET_LOCATION = (SOURCE = (METHOD = HSM))
```

2. Start the sqlplus session to connect to PDB.

```
$ sqlplus / as sysdba

SQL> Connect <pdb_admin>/<pdb_admin_password>@Pluggable Database Service name
```

For Example:

```
SQL> connect salesadm/temp123#@salespdb

Connected
```

3. Run the ADMINISTER KEY MANAGEMENT SQL statement using the following syntax:

```
SQL> ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN IDENTIFIED BY “hsm_partition_password";
```

For example:

```
SQL> ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN IDENTIFIED BY "userpin1";

keystore altered.
```

NOTE: Please make sure that keystore for CDB (root container) is opened and Master key for CDB is generated before opening the keystore and generating the Master key for PDB. Also do not configure HSM auto login for CDB until you generate the master key for PDB (all PDB in case multiple PDB are using the TDE). After generating the Master key for all PDBs you can configure the CDB for auto login and it will work for all PDBs as well.

4. Run the following SQL statement to create the PDB Master key:

```
SQL> ADMINISTER KEY MANAGEMENT SET KEY IDENTIFIED BY "hsm_partition_password";
```

For example:

```
SQL> ADMINISTER KEY MANAGEMENT SET KEY IDENTIFIED BY "userpin1";
```

```
keystore altered.
```

This will `generate` a new master key and from now onwards any encryption/decryption operations performed within this pdb will use this master key. For example execute the following queries:

5. Create a CUSTOMERS table in the PDB.

```
SQL> CREATE TABLE CUSTOMERS (ID NUMBER(5), NAME VARCHAR(42), CREDIT_LIMIT NUMBER(10));
```

6. Enter some values in the CUSTOMERS table.

```
SQL> INSERT INTO CUSTOMERS VALUES (001, 'George Bailey', 10000);

SQL> INSERT INTO CUSTOMERS VALUES (002, 'Denial Vettory', 20000);
```

7. Encrypt the 'CREDIT_LIMIT' column of the 'CUSTOMERS' table:

```
SQL> ALTER TABLE CUSTOMERS MODIFY (CREDIT_LIMIT ENCRYPT);
```

8. With the next command, the values listed in the encrypted column are returned in clear text; Transparent Data Encryption decrypts them automatically:

```
SQL> SELECT CREDIT_LIMIT FROM CUSTOMERS;
```

9. The next command lists encrypted columns in your databases:

```
SQL> SELECT * FROM DBA_ENCRYPTED_COLUMNS;
```

10. Create an encrypted tablespace:

```
SQL> CREATE TABLESPACE SECURESPACE DATAFILE '+DATA' SIZE 150M ENCRYPTION DEFAULT STORAGE
(ENCRYPT);
```

11. Create a table in the tablespace:

```
SQL> CREATE TABLE EMPLOYEE (ID NUMBER(5),NAME VARCHAR(42),SALARY NUMBER(10)) TABLESPACE
SECURESPACE;
```

12. Insert some values in EMPLOYEE table:

```
SQL> INSERT INTO EMPLOYEE VALUES (001,'JOHN SMITH',15000);

SQL> INSERT INTO EMPLOYEE VALUES (002,'SCOTT TIGER',25000);

SQL> INSERT INTO EMPLOYEE VALUES (003,'DIANA HAYDEN',35000);
```

13. Display the contents of the EMPLOYEE table with the following command:

```
SQL> SELECT * FROM EMPLOYEE;
```

14. Close the keystore:

```
SQL> SQL> ADMINISTER KEY MANAGEMENT SET KEYSTORE CLOSE IDENTIFIED BY “hsm_partition_password”;
```

15. After closing the keystore execute the command to display the contents again:

```
SQL> SELECT * FROM EMPLOYEE;
```

You will get the following error that means you cannot list the contents of EMPLOYEE table, if keystore is closed.

```
ERROR at line 1:

ORA-28365: wallet is not open
```

16. Open the keystore:

```
SQL> ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN IDENTIFIED BY “hsm_partition_password”;
```

The above queries are same as we did for root container but it will use the Master Encryption Key of PDB generated on HSM. After generating the Master key on any node it will be accessed by all nodes.

This chapter completes Oracle 12c RAC Integration with SafeNet Luna HSM and demonstrates all the scenarios listed in the beginning of chapter.

7

Integrating SafeNet Luna HSM with Oracle Database 11g R2 RAC

Understanding Oracle RAC

A cluster comprises multiple interconnected computers or servers that appear as if they are one server to end users and applications. Oracle RAC enables you to cluster an Oracle database. Oracle RAC uses Oracle Clusterware for the infrastructure to bind multiple servers so they operate as a single system.

Oracle Clusterware is a portable cluster management solution that is integrated with Oracle Database. Oracle Clusterware is also a required component for using Oracle RAC. In addition, Oracle Clusterware enables both non-cluster Oracle databases and Oracle RAC databases to use the Oracle high-availability infrastructure. Oracle Clusterware enables you to create a clustered pool of storage to be used by any combination of non-cluster and Oracle RAC databases

Non-cluster Oracle databases have a one-to-one relationship between the Oracle database and the instance. Oracle RAC environments, however, have a one-to-many relationship between the database and instances. Oracle RAC databases differ architecturally from non-cluster Oracle databases in that each Oracle RAC database instance also has:

- At least one additional thread of redo for each instance
- An instance-specific undo tablespace

The combined processing power of the multiple servers can provide greater throughput and Oracle RAC scalability than is available from a single server.

Oracle Database RAC Setup

You should familiarize yourself with Oracle Database RAC. Refer to the Oracle Database 11g R2 RAC documentation for more information to install and pre-installation requirements.

The two machines utilized are denoted in the setup as follows:

- RAC1.localdomain
- RAC2.localdomain

Supported Platforms

The following platforms are supported for Luna HSM:

Operating System	SafeNet Luna HSM	Oracle Database Software
Red Hat Enterprise Linux 5.8 (64 bit)	Luna SA v5.2.1 f/w 6.10.1	11.2.0.3
Red Hat Enterprise Linux 6.0 (64 bit)	Luna SA v5.1 f/w 6.2.1	11.2.0.3

Setting up SafeNet Luna HSM for Transparent Data Encryption with Oracle RAC

Verifying Oracle RAC Installation

Before proceeding for HSM based wallet management, it is assumed that Oracle RAC is setup properly and running at this point, you can verify the RAC running information by executing the following commands on any RAC instances:

```
# srvctl config database -d RAC
Database unique name: RAC
Database name: RAC
Oracle home: /u01/app/oracle/product/11.2.0/db_1
Oracle user: oracle
Spfile: +DATA/RAC/spfileRAC.ora
Domain: localdomain
Start options: open
Stop options: immediate
Database role: PRIMARY
Management policy: AUTOMATIC
Server pools: RAC
Database instances: RAC1,RAC2
Disk Groups: DATA
Services:
Database is administrator managed

# srvctl status database -d RAC
Instance RAC1 is running on node rac1
Instance RAC2 is running on node rac2
```

The V$ACTIVE_INSTANCES view can also display the current status of the instances.

```
---------------------------------------------------------------------------------------------------
---------------
$ sqlplus / as sysdba

SQL*Plus: Release 11.2.0.3.0 Production on Mon Oct 14 13:31:35 2013

Copyright (c) 1982, 2011, Oracle.  All rights reserved.


Connected to:
```

```
Oracle Database 11g Enterprise Edition Release 11.2.0.3.0 - 64bit Production
With the Partitioning, Real Application Clusters, Automatic Storage Management, OLAP,
Data Mining and Real Application Testing options

SQL> SELECT inst_name FROM v$active_instances;


INST_NAME
------------------------------------------------------------
rac1.localdomain:RAC1
rac2.localdomain:RAC2
------------------------------------------------------------
```

Configuring the PKCS11 Provider on Oracle RAC Instances

To set up Luna HSM for TDE with Oracle RAC, kindly perform the following steps on RAC1 and RAC2:

Oracle requires the PKSC#11 library provided by HSM vendor. Copy the Luna SA PKCS#11 library to the specified directory structure recommended by Oracle to ensure that the database is able to find this library. Use the following directory structures for UNIX and Windows respectively:

/opt/oracle/extapi/[32,64]/hsm/{VENDOR}/{VERSION}/libapiname.ext

%SYSTEM_DRIVE%\oracle\extapi\[32,64]\hsm\{VENDOR}\{VERSION}\libapiname.ext

Here:

- [32,64] specifies whether the supplied binary is 32-bits or 64-bits
- VENDOR stands for the name of the vendor supplying the library
- VERSION refers to the version of the library. This should preferably be in a format, number.number.number
- apiname requires no special format. However, the apiname must be prefixed with the word lib, as illustrated in the syntax.
- .ext needs to be replaced by the extension of the library file. This extension is .so on Unix.

NOTE: Only one PKCS#11 library is supported at a time. Oracle user should have the read/write permission of the above directory.

For example,
/opt/oracle/extapi/64/hsm/safenet/5.2.1/libshim.so (Linux)

If you are using Luna SA v5.0 and above on Red Hat Enterprise Linux or Oracle Linux then you need to do the following changes in the /etc/Chrystoki.conf file:

- Provide the reference of libshim library in the Crystoki2 section.
- Add the Shim2 section that refers the LibCryptoki library.

For Example:

64 bit Client:

```
Chrystoki2 = {
LibUNIX64=/opt/oracle/extapi/64/hsm/safenet/5.4.0/libshim.so;
}
Shim2 = {
```

```
LibUNIX64=/usr/lib/libCryptoki2_64.so;
}
```

32 bit Client:

```
Chrystoki2 = {
LibUNIX=/opt/oracle/extapi/64/hsm/safenet/5.4.0/libshim.so;
}
Shim2 = {
LibUNIX=/usr/lib/libCryptoki2.so;
}
```

Logged on as oracle user and export the following variables:

```
export ORACLE_SID=orcl
export ORACLE_BASE=/u01/app/oracle                  (oracle installation directory)
export ORACLE_HOME=$ORACLE_BASE/product/11.2.0/dbhome_1
export PATH=$PATH:$ORACLE_HOME/bin
export TNS_ADMIN=$ORACLE_HOME/network/admin
```

Migrating Master Encryption Key from software wallet to HSM

In order to migrate a Master Encryption Key for HSM-Based Encryption, perform the following instructions:

NOTE: Assuming that no software-based wallet is yet created in the directory you would specify to create one.

Verifying that the 'traditional' software-based wallet is working fine

1. Create the directory /etc/oracle/wallet/RAC and permit the oracle user to access this directory on both RAC1 and RAC2:

```
# mkdir -pv /etc/oracle/wallet/RAC

# cd /etc

# chown -R oracle:oinstall oracle/wallet/

# chmod -R 700 oracle/wallet/
```

NOTE: Create the identical directory for storing wallet on both RAC instances or you can use the shared disk for storing the wallet. It will ease our work of copying the wallet manually on all instances. You can use the ASMCA utility to create the ACFS (ASM Cluster File Systems) file and mount this file on disk that will be used by all instances. You can follow the oracle documentation for creating the ACFS file for storing wallet on clustered system.

2. Create or add the following to your $ORACLE_HOME/network/admin/sqlnet.ora – file on both instances of RAC (for e.g. RAC1 and RAC2):
```
ENCRYPTION_WALLET_LOCATION = (SOURCE = (METHOD = FILE) (METHOD_DATA = (DIRECTORY = /etc/oracle/wallet/RAC)))
```
3. Start the database on any node (i.e. RAC1):
```
$ sqlplus / as sysdba
```
Connect to the database as 'system':
```
SQL> connect system/<password>
```
4. Create an encryption wallet; the master key is added into it automatically; the double quotes are mandatory:
```
SQL> alter system set encryption key identified by "wallet_password";
```
Copy the ewallet.p12 file created in the directory /etc/oracle/wallet/RAC from RAC1 to RAC2 in the same directory on RAC2.
5. Encrypt the 'credit_limit' column of the 'CUSTOMERS' table which is owned by the user 'OE':
```
SQL> alter table oe.customers modify (credit_limit encrypt);
```
6. With the next command, the values listed in the encrypted column are returned in clear text; Transparent Data Encryption decrypts them automatically:
```
SQL> select credit_limit from oe.customers where rownum <15;
```
7. The next command lists encrypted columns in your database:
```
SQL> select * from dba_encrypted_columns;
```
8. Finally, this view contains information about the wallet itself:
```
SQL> select * from gv$encryption_wallet;
```
9. Create an encrypted tablespace in the shared disk.
```
SQL> CREATE TABLESPACE mytablespace DATAFILE '+DATA' SIZE 150M ENCRYPTION DEFAULT STORAGE (ENCRYPT);
```
10. Create a table in the TABLESPACE
```
SQL> create table employee (id number(5), name varchar(42), salary number(10)) TABLESPACE mytablespace;
```
11. Insert some values in EMPLOYEE table.
```
SQL> Insert into employee values (001,'JOHN SMITH',10000);

SQL> Insert into employee values (002,'SCOTT TIGER',20000);

SQL> Insert into employee values (003,'DIANA HAYDEN',50000);
```
12. Display the contents of the EMPLOYEE table with the following command:
```
SQL> select * from employee;
```
13. Close the wallet
```
SQL> alter system set encryption wallet close identified by "hsm_partition_pwd";
```
14. Now try to access the contents of EMPLOYEE table
```
SQL> select * from employee;
```
You will get the following error that means you cannot list the contents of EMPLOYEE table, if wallet is closed.
```
ERROR at line 1:
ORA-28365: wallet is not open
```
Now you can go on RAC 2 machine.

15. Firstly open the wallet on RAC2 machine

```
SQL> alter system set encryption wallet open identified by “hsm_partition_pwd”;
```

16. Now try to access the contents of EMPLOYEE table

```
SQL> select * from employee;
```

17. Close the wallet:

```
SQL> alter system set encryption wallet close identified by “wallet_password”;
SQL> exit
```

Now onwards when you start the database you need to open the HSM based wallet to view the encrypted data. You can set the HSM based wallet to Auto-Login that will automatically open when database starts

NOTE: Above commands works for both RAC1 and RAC2 after generating the master key on any instance and copying the wallet on all other instances.

Test the Migration of Software Wallet to HSM Device

1. Change the FILE to HSM in your $ORACLE_HOME/network/admin/sqlnet.ora – file on both instances of RAC (for e.g. RAC1 and RAC2):

```
ENCRYPTION_WALLET_LOCATION = (SOURCE = (METHOD = HSM) (METHOD_DATA = (DIRECTORY =
/etc/oracle/wallet/RAC)))
```

2. Start the database:

```
$ sqlplus / as sysdba
```

Connect to the database as ‘system’:

```
SQL> connect system/<password>
```

3. Execute the following query to migrate the wallet onto the HSM device:

```
SQL> alter system set encryption key identified by “hsm_partition_pwd” migrate using
“wallet_password;
```

“hsm_partition_pwd” is the password for the HSM partition where the Master Encryption Key would be generated.

Copy the ewallet.p12 file created in the directory /etc/oracle/wallet/RAC from RAC1 to RAC2 in the same directory on RAC2.

4. With the next command, the values listed in the encrypted column are returned in clear text; Transparent Data Encryption decrypts them automatically:

```
SQL> select credit_limit from oe.customers where rownum <15;
```

5. The next command lists encrypted columns in your database:

```
SQL> select * from dba_encrypted_columns;
```

6. Finally, this view contains information about the wallet itself:

```
SQL> select * from gv$encryption_wallet;
```

7. Close the wallet:

```
SQL> alter system set encryption wallet close identified by “hsm_partition_pwd”;
SQL> exit
```

NOTE: Above commands works for both RAC1 and RAC2 after copying the wallet on both instances.

8. You can change the Software wallet password to HSM partition password .You can give the following command

```
# orapki wallet change_pwd –wallet [wallet_location/ewallet.p12] –oldpwd <software wallet password > -newpwd <HSM partition password>
```

9. Start the database:

```
$ sqlplus / as sysdba
```

Connect to the database as 'system':

```
SQL> connect system/<password>

SQL> alter system set encryption wallet open identified by "hsm_partition_pwd"
```

This opens both the HSM and the software wallet.

10. You can see the wallets status by executing the following:

```
SQL> select * from gv$encryption_wallet;
```

11. Close the wallet:

```
SQL> alter system set encryption wallet close identified by "wallet_password";
SQL> exit
```

NOTE: Above commands works for both RAC1 and RAC2 after changing the wallet password on both instances. You can copy the wallet from one RAC instance to other after changing the wallet password on an instance.

Create the auto-login wallet on both the instances RAC1 and RAC2. Change the password (if you wish to) back to the initial password for software based wallet using orapki and create an auto-login software based wallet.

```
# cd <path to the oracle wallet directory>
# orapki wallet create -wallet . -auto_login
```

When prompt for password, provide the software wallet password.

Verify that an auto-open software wallet has been created in the oracle wallet directory you specified in the sqlnet.ora file: You will find two wallets in this directory: "ewallet.p12" and "cwallet.sso"; the latter is the auto-open wallet;

Oracle selects encryption wallet first and then auto wallet. Rename or move the encryption wallet from the location you have specified in sqlnet.ora file:

```
# cd /etc/oracle/wallet/RAC
# mv ewallet.p12 ewallet.p24
```

It will prevent the Transparent Data Encryption to open it.

NOTE: Rename the encryption wallet on both instances of RAC to make the local wallet to auto-login. You can copy the auto-login wallet and encryption wallet from one RAC instance to other after renaming the encryption wallet.

Connect to the database as system and open the HSM wallet (the software wallet is already open):

```
SQL> alter system set encryption wallet open identified by “hsm_partition_pwd”;
```

Now onwards when you start the database you need to open the HSM wallet only, local wallet remains open automatically.

Generating Master Encryption Key Directly on HSM

In order to generate a Master Encryption Key for HSM-Based Encryption, perform the following instructions:

NOTE: It is assumed that no traditional wallet is generated and database is not using TDE.

1. Create or add the following to your $ORACLE_HOME/network/admin/sqlnet.ora – file on both instances of RAC (for e.g. RAC1, RAC2):

   ```
   ENCRYPTION_WALLET_LOCATION = (SOURCE = (METHOD = HSM))
   ```

2. Start the database:

   ```
   $ sqlplus / as sysdba
   ```

   If the database is not yet started, you can start it using:

   ```
   SQL> startup;
   ```
   Connect to the database as ‘system’:
   ```
   SQL> connect system/<password>
   ```

3. Create an encryption wallet. The master key would automatically be created onto the HSM.

   ```
   SQL> alter system set encryption key identified by “hsm_partition_pwd”;
   ```
   To use multiple slots or partitions, the syntax to generate master key to a slot or partition, use the following syntax:
   ```
   SQL> alter system set encryption key identified by "hsm_partition_pwd|<partition_name>";
   ```

   NOTE: If you have configured the HA on Luna SA then you do not need to provide the partition_name. In HA key would be generated on both partitions, above command would be use when you have registered multiple SA partitions and want to use a single partition to generate the keys.

   You can verify the generated keys on HSM.

   Here is snapshot from the HSM

   ```
   Partition Name:  part3
    Partition SN:    150207010
    Storage (Bytes): Total=102701, Used=348, Free=102353
   ```

```
Number objects:  2

Object Label:  ORACLE.TDE.HSM.MK.068263CF3494A14F26BF17D57D4D080333
Object Type:   Symmetric Key

Object Label:  ORACLE.TSE.HSM.MK.078C5EF205FE25A59D6B999E14265F031E0203
Object Type:   Symmetric Key
```

4. Encrypt the 'credit_limit' column of the 'CUSTOMERS' table which is owned by the user 'OE':

```
SQL> alter table oe.customers modify (credit_limit encrypt);
```

5. With the next command, the values listed in the encrypted column are returned in clear text; Transparent Data Encryption decrypts them automatically:

```
SQL> select credit_limit from oe.customers where rownum <15;
```

6. The next command lists encrypted columns in your database:

```
SQL> select * from dba_encrypted_columns;
```

7. Finally, this view contains information about the wallet itself.

```
SQL> select * from gv$encryption_wallet;
```

8. Create an encrypted tablespace in the shared disk.

```
SQL> CREATE TABLESPACE mytablespace DATAFILE '+DATA' SIZE 150M ENCRYPTION DEFAULT STORAGE
(ENCRYPT);
```

9. Create a table in the TABLESPACE

```
SQL> create table employee (id number(5), name varchar(42), salary number(10)) TABLESPACE
mytablespace;
```

10. Insert some values in EMPLOYEE table.

```
SQL> Insert into employee values (001,'JOHN SMITH',10000);
SQL> Insert into employee values (002,'SCOTT TIGER',20000);
SQL> Insert into employee values (003,'DIANA HAYDEN',50000);
```

11. Display the contents of the EMPLOYEE table with the following command:

```
SQL> select * from employee;
```

12. Close the wallet.

```
SQL> alter system set encryption wallet close identified by “hsm_partition_pwd”;
```

13. Now try to access the contents of EMPLOYEE table

```
SQL> select * from employee;
```

    You will get the following error that means you cannot list the contents of EMPLOYEE table, if wallet is closed.

```
ERROR at line 1:
ORA-28365: wallet is not open
```

    Now go to RAC 2 machine.

14. Firstly open the wallet on RAC2 machine.

```
SQL> alter system set encryption wallet open identified by “hsm_partition_pwd”;
```

15. Now try to access the contents of EMPLOYEE table.

```
SQL> select * from employee;
```

Now onwards when you start the database you need to open the HSM based wallet to view the encrypted data. You can set the HSM based wallet to Auto-Login that will automatically open when database starts

Setting up Oracle to Create Auto-Open Wallet

We have categorized the possible cases on which we can create the Auto-Open HSM wallet, use the steps according to the cases described below:

1. When TDE is used in 'HSM only' mode (never migrated from an Oracle Wallet):

a. The current entry in sqlnet.ora:

```
ENCRYPTION_WALLET_LOCATION = (SOURCE = (METHOD = HSM))
```

Needs to be changed to:

```
ENCRYPTION_WALLET_LOCATION = (SOURCE = (METHOD = HSM) (METHOD_DATA = (DIRECTORY =
/etc/oracle/wallet/RAC)))
```

b. Create a (local) auto-open and encryption wallet in /etc/oracle/wallet/RAC:

```
# cd /etc/oracle/wallet/RAC

# orapki wallet create -wallet . -auto_login
```

NOTE: When prompt for password you need to provide the password of at least 8 characters. It will be your software wallet password.

c. Add the following entry to the empty wallets to enable an 'auto-open' HSM:

```
# mkstore -wrl . -createEntry ORACLE.TDE.HSM.AUTOLOGIN <any-non-empty-string>
```

NOTE: <any-non-empty-string> could be any string of alphanumeric characters and when asked for password, provide the software wallet password.

d. By default oracle choose the encryption wallet and if it is not available it will choose the auto wallet. Rename the encryption wallet (ewallet.p12) or move it out of the 'ENCRYPTION_WALLET_LOCATION' defined in 'sqlnet.ora' to a secure location; do not delete the encryption wallet and do not forget the wallet password.

NOTE: Rename the ewallet.p12 to ewallet.p24 and copy both ewallet.p24 and cwallet.sso files created in the directory /etc/oracle/wallet/RAC from RAC1 to RAC2 in the same directory on RAC2 after executing the above command and make the same changes for sqlnet.ora file.

e. Close the connection to the HSM with

```
SQL> alter system set encryption wallet close identified by "Partition_password";
```

And open it one last time with

```
SQL> alter system set encryption wallet open identified by “Partition_password”;
```

This will insert “Partition_password” into the auto-open wallet; from now on, no password is required to access encrypted data with the TDE master encryption key stored in an HSM. You can also access the encrypted tablespace data.

2. When TDE was never used before (you are using the TDE first time and want to create HSM auto login):

a. Creat`e a new entry in sqlnet.ora:

```
ENCRYPTION_WALLET_LOCATION = (SOURCE = (METHOD = HSM) (METHOD_DATA = (DIRECTORY =
/etc/oracle/wallet/RAC)))
```

b. Create a (local) auto-open and encryption wallet in /etc/oracle/wallet/RAC:

```
# orapki wallet create -wallet . -auto_login
```

NOTE: When prompt for password you need to provide the password of at least 8 characters. It will be your software wallet password.

c. Add the following entry to the empty wallets to enable an ‘auto-open HSM’:

```
# mkstore -wrl . -createEntry ORACLE.TDE.HSM.AUTOLOGIN <any-non-empty-string>
```

NOTE: <any-non-empty-string> could be any string of alphanumeric characters and when asked for password, provide the software wallet password.

d. By default oracle choose the encryption wallet and if it is not available it will choose the auto wallet. Rename the encryption wallet (ewallet.p12) or move it out of the ‘ENCRYPTION_WALLET_LOCATION’ defined in ‘sqlnet.ora’ to a secure location; do not delete the encryption wallet and do not forget the wallet password.

NOTE: Rename the ewallet.p12 to ewallet.p24 and copy both ewallet.p24 and cwallet.sso files created in the directory /etc/oracle/wallet/RAC from RAC1 to RAC2 in the same directory on RAC2 after executing the above command and make the same changes for sqlnet.ora file.

e. Create a TDE master encryption key inside the HSM:

```
SQL> alter system set encryption key identified by “Partition_password”
```

This will insert “Partition_password into the auto-open wallet; from now on, no password is required to access encrypted data with the TDE master encryption key stored in an HSM.

3. When an HSM and an encryption wallet is in use:

a. At the end of a prior migration from Oracle Wallet to HSM, the wallet password was changed to the "Partition_password"; sqlnet.ora contains the following entry:

```
ENCRYPTION_WALLET_LOCATION = (SOURCE = (METHOD = HSM) (METHOD_DATA =
(DIRECTORY=/etc/oracle/wallet/RAC)))
```

b. An auto-open HSM requires the wallet to be a (local) auto-open wallet as well; this (local) auto-open wallet will contain the TDE master key from the encryption wallet and the auto-open string for the HSM.

Create an auto-open wallet from the encryption wallet:

```
# orapki wallet create -wallet . -auto_login
```

c. Continue at 4.c.)

4. When an HSM and a (local) auto-open wallet is in use:

a. At the end of a prior migration from Oracle Wallet to HSM, the wallet password was not changed to the "Partition_password"; a (local) auto-open wallet is used instead and the encryption wallet was either renamed or removed from the "ENCRYPTION_WALLET_LOCATION"; sqlnet.ora contains the following entry:

```
ENCRYPTION_WALLET_LOCATION = (SOURCE = (METHOD = HSM) (METHOD_DATA = (DIRECTORY
=/etc/oracle/wallet/RAC)))
```

b. Restore the encryption wallet from its secure location or rename it back to the original file name ewallet.p12

c. Add the following entry to the wallets to enable an 'auto-open HSM', applying the wallet password for the encryption wallet:

```
# mkstore -wrl . -createEntry ORACLE.TDE.HSM.AUTOLOGIN <any-non-empty-string>
```

d. Rename the encryption wallet or move it out of the ENCRYPTION_WALLET_LOCATION defined in sqlnet.ora, do not delete the encryption wallet and do not forget the wallet password.

NOTE: Rename the ewallet.p12 to ewallet.p24 and copy both ewallet.p24 and cwallet.sso files created in the directory /etc/oracle/wallet/RAC from RAC1 to RAC2 in the same directory on RAC2 after executing the above command and make the same changes for sqlnet.ora file.

e. Close the connection to the HSM with

```
SQL> alter system set encryption wallet close identified by "Partition_password";
```

And open it one last time with

```
SQL> alter system set encryption wallet open identified by "Partition_password";
```

This will automatically insert "Partition_password" into the auto-open wallet. From now on, no password is required to access encrypted data with the TDE master encryption key stored in an HSM.

5. When only an encryption wallet is in use (no HSM):

a. sqlnet.ora contains this entry:

```
ENCRYPTION_WALLET_LOCATION = (SOURCE = (METHOD = FILE) (METHOD_DATA = (DIRECTORY
=/etc/oracle/wallet/RAC)))
```

b. An auto-open HSM requires the wallet to be a (local) auto-open wallet as well; this (local) auto-open wallet will contain the TDE master key from the encryption wallet and the auto-open string for the HSM. Create an auto-open wallet from the encryption wallet:

```
# orapki wallet create –wallet . -auto_login
```

c. Continue at 6.b.)

6. When a (local) auto-open wallet is in use:

a. Add the following entry to the wallets to enable an 'auto-open HSM':

```
# mkstore -wrl . -createEntry ORACLE.TDE.HSM.AUTOLOGIN <any-non-empty-string>
```

b. Change the entry in sqlnet.ora to:

```
ENCRYPTION_WALLET_LOCATION = (SOURCE = (METHOD = HSM) (METHOD_DATA = (DIRECTORY
=/etc/oracle/wallet/RAC)))
```

c. Migrate the TDE column encryption master key from wallet to HSM with:

```
SQL> alter system set encryption key identified by "Partition_password" migrate using
"wallet_password";
```

This will insert "Partition_password into the auto-open wallet. From now on, no password is required to access encrypted data with the TDE master encryption key stored in an HSM.

d. Rename the encryption wallet or move it out of the ENCRYPTION_WALLET_LOCATION defined in sqlnet.ora; do not delete the encryption wallet and do not forget the wallet password.

NOTE: Rename the ewallet.p12 to ewallet.p24 and copy both ewallet.p24 and cwallet.sso files created in the directory /etc/oracle/wallet/RAC from RAC1 to RAC2 in the same directory on RAC2 after executing the above command and make the same changes for sqlnet.ora file.

8
Troubleshooting Tips

Problem – 1

Error message “ORA-43000: PKCS11: library not found” or “ORA-28376: cannot find PKCS11 library” when trying to generate Master Key on HSM or migrate keys from wallet to HSM.

Solution:

1. Make sure that library path is set correctly, for example:

   /opt/oracle/extapi/[32/64]/HSM/[x.x.x]/libcryptoki2_64.so

2. Ensure that oracle : oinstall is the owner : group of this directory with read/write permissions.
3. Make sure that 64-bit JVM is running on the machine on which we are using 64 bit client because 32 bit JVM would not able to use the 64 bit library.

Problem – 2

Error message in PDB database “ORA-46627: keystore password mismatch” when trying to open the keystore or generating Master Key on HSM.

Solution:

1. Make sure that the provided HSM password is correct.
2. Ensure that HSM Auto_Login or Local_Auto_Login is not enabled for the CDB, if it enabled then please use encryption wallet instead of auto wallet in CDB then perform this operation in PDB. After generating the Master Key for PDB you can enable the Auto_Login again in CDB and it works for all PDBs as well.