SafeNet PKCS#11 with Oracle TDE

Integration Guide

gemalto
security to be free

Document Part Number: 007-013430-001, Rev B

Release Date: March 2017

Contents

Preface

Audience

This document is intended for personnel responsible for maintaining your organization's security infrastructure. This includes SafeNet KeySecure users and security officers, the key manager administrators, and network administrators. It is assumed that the users of this document are proficient with security concepts.

All products manufactured and distributed by Gemalto are designed to be installed, operated, and maintained by personnel who have the knowledge, training, and qualifications required to safely perform the tasks assigned to them. The information, processes, and procedures contained in this document are intended for use by trained and qualified personnel only.

Document Conventions

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

Notifications

This template uses notes, cautions, and warnings to alert you to important information that may help you to complete your task, or prevent personal injury, damage to the equipment, or data loss.

Notes

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

NOTE: Take note. Notes contain important or helpful information that you want to make stand out to the user.

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**.) • 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.)
Double quote marks	Double quote marks enclose references to other sections within the document. For example: Refer to “Notifications” on page 7.
<variable>	In command descriptions, angle brackets represent variables. You must substitute a value for command line arguments that are enclosed in angle brackets.
[optional] [<optional>] [a | b | c] [<a> | <b> | <c>]	Square brackets enclose optional keywords or <variables> in a command line description. Optionally enter the keyword or <variable> that is enclosed in square brackets, if it is necessary or desirable to complete the task. Square brackets enclose optional alternate keywords or variables in a command line description. Choose one command line argument enclosed within the braces, if desired. Choices are separated by vertical (OR) bars.
{ a | b | c } { <a> | <b> | <c> }	Braces enclose required alternate keywords or <variables> in a command line description. You must choose one command line argument enclosed within the braces. Choices are separated by vertical (OR) bars.

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://safenet.gemalto.com/technical-support/ 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
Overview

This document contains the necessary information to install, configure, and integrate the Oracle Database Transparent Data Encryption (TDE) with SafeNet's KeySecure solution.

Oracle Transparent Data Encryption (TDE) provides the infrastructure necessary for implementing encryption within the database. It enables the organizations to encrypt sensitive application data such as credit card numbers on storage media completely transparent to the application (table columns or table spaces). It encrypts the data in the datafiles so that in case they are obtained by other parties it is not possible to access the clear text data. In the databases where TDE is configured, any user who has access on an encrypted table, can see the data in clear text because Oracle transparently decrypts the data for any user having the necessary privileges.

TDE uses a two-tier encryption key architecture consisting of:

- a master encryption key that is used to encrypt secondary keys used for column encryption and tablespace encryption.
- one or more table and/or tablespace keys. These keys are used to encrypt one or more specific columns or the keys used to encrypt tablespaces. There is only one table key regardless of the number of encrypted columns in a table and it is stored in the data dictionary. The tablespace key is stored in the header of each datafile of the encrypted tablespace.

NOTE: Creation and labelling of TDE keys is governed by Oracle and KeySecure doesn't control it.

The table and tablespace keys are encrypted using the master key. The master key is stored in an External Security Module (ESM) that can be one of the following:

- An Oracle Wallet - a secure container outside of the database. It is encrypted with a password.
- A KeySecure - a device used to secure keys and perform cryptographic operations. Oracle interfaces to the device using a PKCS#11 library supplied by the KeySecure vendor.

The SafeNet KeySecure provides a secure location for storing the TDE master encryption key. SafeNet PKCS#11 provides an industry-standard interface that enables the Oracle database to communicate with the SafeNet KeySecure.

Prerequisites

Oracle Database

Oracle Database 11g (or higher) must be installed on the target machine to carry on with the integration process. Refer to the Oracle documentation for detailed installation instructions for the database and the Transparent Data Encryption feature.

IMPORTANT: It is recommended that the following mandatory patches are installed on Oracle 11.2.0.2 before configuring Oracle TDE with SafeNet PKCS#11:

- RHEL/AIX/ Solaris – Oracle mandatory patch 12626642.
- Windows – Oracle mandatory patch 11.2.0.2 bundle 15 (tracking patch numbers 13413154 (32bit) and 13413155 (64bit)).

For details and exact versions, see “Oracle TDE Platform Matrix” on page 127.

After applying the mandatory patch, make sure to restart the database by executing the `shutdown immediate and startup` commands.

SafeNet KeySecure

The SafeNet KeySecure must be installed and configured to allow authenticated clients the ability to create keys. To enable this, select the **Allow Key and Policy Configuration Operations** check box on the NAE Server Configuration page.

On SafeNet KeySecures running version 5.2.1 or older, the Allow Key and Policy Configuration Operations and Allow Key Export check boxes can be found on the NAE Server Configuration page (**Device >> NAE Server**). On SafeNet KeySecures running version 5.3 or later, these check boxes can be found on the Cryptographic Key Server Configuration page (**Device >> Key Server**. Select **NAE-XML**.)

The SafeNet KeySecure’s IP address and port are required when configuring SafeNet PKCS#11.

Supported Platforms

SafeNet PKCS#11/Oracle TDE is supported on the following platforms:

- Windows Server 2003 32-bit, Windows Server 2008 32-bit
- Windows Server 2003 64-bit, Windows Server 2008 64-bit, Windows Server 2008 R2 64-bit, Windows Server 2012 R2 64-bit
- RHEL 4 64-bit, RHEL 5 64-bit, RHEL 6 64-bit, RHEL 7 64-bit
- AIX 5.3 PowerPC 64-bit, AIX 6.1 PowerPC 64-bit, AIX 7.1 PowerPC 64-bit
- Solaris 10 SPARC 64-bit
- openSUSE 11.2, openSUSE 11.4
- SLES-12

Document PN: 007-013430-001, Rev B, © Gemalto 2015 - 2017,

2
Configuring SafeNet PKCS#11 on Linux/UNIX

Configuration of SafeNet PKCS#11 with Oracle TDE on Linux/UNIX involves the following steps:

1. **Installing the Application Server License on SafeNet KeySecure**
2. **Creating Oracle TDE Authentication Credentials**
3. **Installing SafeNet PKCS#11 Library**
4. **Configuring Connection to SafeNet KeySecure**

Installing the Application Server License on SafeNet KeySecure

To install the Application Server License on SafeNet KeySecure:

1. Obtain an Application Server License file from SafeNet.
2. Install the license file on the SafeNet KeySecure.

   To install the license file:

   a. Log on to the Management Console as an administrator.

   b. Navigate to the System Information page (**Device >> System Information & Upgrade**).

   c. Select the method of installation and click **Upgrade/Install**. The machine will reboot.

Creating Oracle TDE Authentication Credentials

The Oracle database needs a user to authenticate to the SafeNet KeySecure. This user will own the keys generated by Oracle TDE.

To create Oracle TDE authentication credentials on SafeNet KeySecure:

1. Log on to the Management Console as an administrator with Users and Groups access control. (The `admin` account created during the SafeNet KeySecure installation has this access control.)
1. Navigate to the Local Users section of the User & Group Configuration page (**Security >> Local Authentication >> Local Users & Groups**).
2. Click **Add**.

3. Enter a user name in the **Username** field and password in the **Password** field. (This document assumes **tdeowner** as the user name and **asdf1234** as password. If different values are entered here, then adjust the instructions in 5, “Managing HSM Wallets” and 6, “Managing Oracle Wallets” accordingly.)
4. Select the **User Administration Permission and Change Password Permission** check boxes.
5. Click **Save**.

Installing SafeNet PKCS#11 Library

To install SafeNet PKCS#11 library:

1. Download SafeNet PKCS#11 from SafeNet customer support site: https://serviceportal.safenet-inc.com. The software adheres to the following naming convention:

   ```
   SafeNet Part Number - Product Name - Product Version - File Format
   ```

   For example:

   ```
   610-013046-001_pkcs11_tde_linux_64b_v8.3.0.000-0xx.tar.gz
   ```

2. Log on to the Linux/UNIX client as Oracle user.
3. Extract the file using any standard archive utility.

   For example, execute:

   ```
   tar -xzf <source_directory/tar_file_name> -C <destination_directory>
   ```

4. Create the `/opt/oracle/extapi/<ARCH>/hsm/safenet/<VERSION>` directory. The Oracle user must have appropriate access permissions on `/opt/`.

   Where `<ARCH>` is the system architecture (either 32 or 64), and `<VERSION>` is the SafeNet software version number (e.g., 8.3.0).

   From this point onward, this document uses `<ARCH>` as **64** and `<VERSION>` as **8.3.0**. If the system architecture and version is different, adjust these values accordingly.

5. Copy the library file `libIngPKCS11.so-8.3.0.000` from extracted `/SafeNet/PKCS11/lib` directory to `/opt/oracle/extapi/64/hsm/safenet/8.3.0`.

   For example:

   ```
   $ cp libIngPKCS11.so-8.3.0.000 /opt/oracle/extapi/64/hsm/safenet/8.3.0
   ```

   NOTE: The receiving directory is a fixed location. Oracle searches this directory. It cannot be changed. Changing the directory name results in a “`cannot find PKCS11 library`” error.

6. Copy the properties file **IngrianNAE.properties** from extracted `/SafeNet/PKCS11 directory to /opt/oracle/extapi/64/hsm/safenet/8.3.0`.

   For example:

   ```
   $ cp IngrianNAE.properties /opt/oracle/extapi/64/hsm/safenet/8.3.0
   ```

7. Rename libIngPKCS11.so-8.3.0.000 as libIngPKCS11.so.

   For example:

   ```
   $ mv libIngPKCS11.so-8.3.0.000 libIngPKCS11.so.
   ```

Configuring Connection to SafeNet KeySecure

To configure connection to SafeNet KeySecure:

1. Configure the connection to SafeNet KeySecure.

   Enter the following values in the **IngrianNAE.properties** file (placed at `/opt/oracle/extapi/64/hsm/safenet/8.3.0`).

   - **NAE_IP** – IP address of the SafeNet KeySecure.
   - **NAE_Port** – 9000 (This is the default value).
   - **Log_Level** – MEDIUM (This is the default, but it can be set to HIGH for troubleshooting).
   - **Log_File** – Full path and file name. The Oracle user must have write permissions for the path and file. A public location such as `/tmp` is recommended.

2. Modify environment variables for the Oracle user.

   Make sure that the following environment variables are exported so that they are inherited by new Oracle server processes. Edit the shell profile (in many shells, the file is called **.profile** and is located in the home directory of the Oracle user.)

   ```
   export SFNT_HSMAPI_BASE=/opt/oracle/extapi/<32|64>/hsm/safenet/8.3.0

   export NAE_Properties_Conf_Filename=$SFNT_HSMAPI_BASE/IngrianNAE.properties

   export IngrianNAE_Properties_Conf_Slot_ID_Max=100

   export IngrianNAE_Properties_Conf_SessionID_Max=100
   ```

3. Add the environment variable (**AIX and Solaris**).

   - **AIX** – Add the environment variable, LIBPATH, as follows:

     ```
     export LIBPATH=/opt/oracle/extapi/64/hsm/safenet/8.3.0:
     /home/oracle/SafeNet/PKCS11/samplelibs
     ```

   - **Solaris** – Add the environment variable, LD_LIBRARY_PATH, as follows:

     ```
     export LD_LIBRARY_PATH=/opt/oracle/extapi/64/hsm/safenet/8.3.0:
     /home/oracle/SafeNet/PKCS11/samplelibs
     ```

   Here, samplelibs is provided in the SafeNet PKCS#11 TDE package.

3

Configuring SafeNet PKCS#11 on Windows

Configuration of SafeNet PKCS#11 with Oracle TDE on Windows involves the following steps:

1. **Installing the Application Server License on SafeNet KeySecure**
2. **Creating Oracle TDE Authentication Credentials**
3. **Installing SafeNet PKCS#11 Library**
4. **Configuring Connection to SafeNet KeySecure**

Installing the Application Server License on SafeNet KeySecure

To install the Application Server License on SafeNet KeySecure:

1. Obtain an Application Server License file from SafeNet.
2. Install the license file on the SafeNet KeySecure.

   To install the license file:

   a. Log on to the Management Console as an administrator.

   b. Navigate to the System Information page (**Device >> System Information & Upgrade**).

   c. Select the method of installation and click **Upgrade/Install**. The machine will reboot.

Creating Oracle TDE Authentication Credentials

The Oracle database needs a user to authenticate to the SafeNet KeySecure. This user will own the keys generated by Oracle TDE.

To create Oracle TDE authentication credentials on SafeNet KeySecure:

1. Log on to the Management Console as an administrator with Users and Groups access control. (The admin account created during the SafeNet KeySecure installation has this access control.)
2. Navigate to the Local Users section of the User & Group Configuration page (**Security >> Local Authentication >> Local Users & Groups**).
3. Click **Add**.
4. Enter a user name in the **Username** field and password in the **Password** field. (This document assumes **tdeowner** as the user name and **asdf1234** as password. If different values are entered here, then adjust the instructions in CHAPTER 6, “Managing HSM Wallets” and CHAPTER 7, “Managing Oracle Wallets” accordingly.)
5. Select the **User Administration Permission and Change Password Permission** check boxes.
6. Click **Save**.

Installing SafeNet PKCS#11 Library

To install SafeNet PKCS#11 library:

1. Download SafeNet PKCS#11 from SafeNet customer support site: https://serviceportal.safenet-inc.com. The software adheres to the following naming convention:

   ```
   SafeNet Part Number - Product Name - Product Version - File Format
   ```

   For example:

   ```
   610-013046-001_pkcs11_tde_linux_64b_v8.3.0.000-0xx.tar.zip
   ```

2. Unzip the file using any standard archive utility.
3. Double-click the executable file to open the installation wizard.
4. Walk through the wizard to complete the installation. This creates the *PKCS11* directory at *C:\Program Files\Ingrian*.
5. Create the `%SYSTEM_DRIVE%\oracle\extapi\<ARCH>\hsm\safenet\<VERSION>` directory.

   Where `%SYSTEM_DRIVE%` is a drive on the database server (for example, **C:** or **D:**); `<ARCH>` is the system architecture (either 32 or 64); and `<VERSION>` is the SafeNet software version number (e.g., 8.3.0).

   From this point onward, this document uses `<ARCH>` as **64** and `<VERSION>` as **8.3.0**. If the system architecture and version is different, adjust these values accordingly.
6. Copy the **ingPKCS11.dll** file from the *C:\Program Files\Ingrian\PKCS11* folder to `%SYSTEM_DRIVE%\oracle\extapi\64\hsm\safenet\8.3.0`
7. Copy the **IngrianNAE.properties** file from *the C:\Program Files\Ingrian\PKCS11* folder to `%SYSTEM_DRIVE%\oracle\extapi\64\hsm\safenet\8.3.0`
8. Update the location of the **IngrianNAE.properties** file in the registry. Set the value of `HKEY_LOCAL_MACHINE\SOFTWARE\Ingrian\NAE_Properties_Config\ConfigFileName` to `%SYSTEM_DRIVE%\oracle\extapi\64\hsm\safenet\8.3.0\IngrianNAE.properties`.

Configuring Connection to SafeNet KeySecure

Configure the connection to the SafeNet KeySecure by entering the following values in the `%SYSTEM_DRIVE%\oracle\extapi\64\hsm\safenet\8.3.0\IngrianNAE.properties` file.

- **NAE_IP** – IP address of the SafeNet KeySecure.
- **NAE_Port** – 9000 (This is the default value).
- **Log_Level** – MEDIUM (This is the default, but it can be set to HIGH for troubleshooting).
- **Log_File** – Full path and file name. The Oracle user must have write permissions for the path and file.

4
Configuring the Properties File

This chapter lists the contents of the **IngrianNAE.properties** file. The properties file defines, among other things, the IP address, port and protocol of the SafeNet KeySecures to which your client connects.

Editing the Properties File

The values in the properties file are case-sensitive. yes is not YES. tcp is not TCP. Follow the example of the default properties file.

You can comment-out values using #. You'll see that the properties file is delivered with both EdgeSecure_Name and Cipher_Spec commented-out. You may want to use comments to save settings when troubleshooting. For example, you could store commonly used NAE_IP addresses like this:

```
NAE_IP=10.0.0.2
#NAE_IP=10.0.0.3
#NAE_IP=10.0.0.4
```

When editing parameters that use time values, you can use the following abbreviations:

- ms - milliseconds e.g. 4500ms for 4.5 seconds
- s - seconds e.g. 30s for 30 seconds
- m - minutes e.g. 5m for 5 minutes
- h - hours e.g. 10h for 10 hours
- d - days e.g. 2d for 2 days

If you do not include an abbreviation, the default time unit is used. For most time-related values the default is milliseconds.

The Parameters

Once you install the client software, you can customize it to meet the needs of your environment by modifying the properties file.

NOTE: The `Unreachable_Server_Retry_Period` and `Maximum_Server_Retry_Period` parameters are no longer included in the properties file.

The Connection_Retry_Interval and Connection_Timeout parameters continue to function in the same way as earlier.

You can modify the Connection_Timeout parameter to specify an appropriate time for which a client waits to connect to a SafeNet KeySecure before timing out. Specifying a large value for the Connection_Timeout parameter results in delayed switching from one SafeNet KeySecure to another.

To achieve the desired Unreachable_Server_Retry_Period, set the Connection_Timeout value using the following formula:

```
Connection_Timeout = Unreachable_Server_Retry_Period , number of servers in a tier
```

As trying to connect to an unreachable server again will delay the transition to the next tier, the Connection_Timeout value should be large enough to make sure that the server is unreachable.

Likewise, to achieve the desired Maximum_Server_Retry_Period, set the Connection_Timeout value using the following formula:

```
Connection_Timeout = Maximum_Server_Retry_Period , number of servers in all tiers
```

The contents of the file, including the delivered settings but excluding the comments, are shown below:

```
Version=3.1
NAE_IP=
NAE_Port=9000
Protocol=tcp
Use_Persistent_Connections=yes
Size_of_Connection_Pool=300
Connection_Timeout=30000
Connection_Read_Timeout=30000
Connection_Idle_Timeout=600000
Connection_Retry_Interval=600000
Cluster_Synchronization_Delay=100
#EdgeSecure_Name=
CA_File=
Cert_File=
Key_File=
Passphrase=
Log_Level=MEDIUM
Log_File=
Log_Rotation=Daily
Log_Size_Limit=100k
```

Version

The Version parameter indicates the version of the properties file and should not be modified.

NAE_IP.1

The NAE_IP.1 parameter specifies the IP address of the SafeNet KeySecure.

Port

The NAE_Port specifies the port of the SafeNet KeySecure. The default port is 9000.

IMPORTANT: Clients and servers must use the same port.

Protocol

The Protocol specifies the protocol used to communicate between the client and the SafeNet KeySecure.

Possible settings:

- **tcp**
- **ssl** - The ssl option uses TLSv1.2. By default, TLSv1.2 is enabled on all SafeNet KeySecures. If you have disabled the use of TLSv1.2 on your servers, then you cannot establish SSL connections with between your NAE clients and servers.

IMPORTANT: Clients and servers must use the same protocol. If your SafeNet KeySecures are listening for SSL requests, and your clients aren't sending SSL requests, you will run into problems.

TIP: We recommend that you gradually increase security after confirming connectivity between the client and the SafeNet KeySecure. Once you have established a TCP connection between the client and server, it is safe to move on to SSL. Initially configuring a client under the most stringent security constraints can complicate troubleshooting.

Use_Persistent_Connections

The `Use_Persistent_Connections` parameter enables the persistent connections functionality.

Possible settings:

- **yes** - Enables the feature. The client establishes persistent connections with the NAE Servers. This is the default value.
- **no** - Disables the feature. A new connection is made for each connection request. The connection is closed as soon as the client receives the server response.

Size_of_Connection_Pool

The `Size_of_Connection_Pool` parameter is the total number of client-server connections that your configuration could possibly allow. (Not what actually exists at a given moment.)

Possible settings:

- **Any positive integer** - The default is 300.

Connections in the pool can be active or waiting, TCP or SSL. A connection is created as needed, and the pool scales as needed. The pool starts at size 0, and can grow to the value set here. Once the pool is full, new connection requests must wait for an existing connection to close.

Connection pooling is configured on a per-client basis. The size of the pool applies to each *client*; it is not a total value for a SafeNet KeySecure or for a load balancing group. If there are multiple clients running on the same machine, separate connection pools are maintained for each client.

Connection_Idle_Timeout

The `Connection_Idle_Timeout` parameter specifies the amount of time connections in the connection pool can remain idle before the client closes them.

Possible settings:

- **Any positive integer** - The default is 600000ms (10 min).

IMPORTANT: There are *two different connection timeout values*: **one on the SafeNet KeySecure, and one in the properties file**. The value of the timeout in the properties file must be **less than** what is set on the server. This lets the client control when idle connections are closed. Otherwise, the client can maintain a connection that is closed on the server side, which can lead to error.

Connection_Timeout

The `Connection_Timeout` parameter specifies how long the client waits for the TCP connect function before timing out.

Possible settings:

- **0** - disables this setting. The client uses the operating system's connect timeout.
- **Any positive integer** - The default is 60000ms.

Setting this parameter a few hundred ms **less than** the operating system's connection timeout makes connection attempts to a downed server fail faster, and failover happens sooner. If a connection cannot be made before the timeout expires, the server is marked as down and taken out of the rotation.

NOTE: If your client is working with many versions of a key, do not set the **Connection_Timeout** parameter too low. Otherwise the client connection may close before the operation is complete.

Connection_Read_Timeout

The `Connection_Read_Timeout` parameter allows you to control how long the client waits when reading data from a SafeNet KeySecure before determining that it is down. Requests should only time out if the SafeNet KeySecure is physically down (e.g. powered off, or not responding because of misconfiguration).

Possible settings:

- **0** - disables this setting. The client waits indefinitely until the SafeNet KeySecure can be reached. Requests remain outstanding until the client's request is successfully satisfied.
- **Any positive integer** - The default is 7000ms.

The purpose of this parameter is to control how you want the client to react when the SafeNet KeySecure is down. If you want it to time out eventually and return an error back to your application, then you should set this value to an appropriate number of milliseconds to allow for requests to complete in high load and high latency situations.

Connection_Retry_Interval

The `Connection_Retry_Interval` parameter determines how long the client waits before trying to reconnect to a disabled server. If one of the SafeNet KeySecures in a load balanced configuration is not reachable, the client assumes that the server is down, and then waits for the specified time period before trying to connect to it again.

Possible settings:

- **0** - This is the infinite retry interval. Once a server gets disabled, it will be brought back into use only after all servers become disabled.
- **Any positive integer** - The default value is 600000ms (10 minutes).

Cluster_Synchronization_Delay

The `Cluster_Synchronization_Delay` parameter specifies how long the client will wait before assuming that key changes have been synchronized throughout a cluster. After creating, cloning, importing, or modifying a key, the client will continue to use the same SafeNet KeySecure appliance until the end of this delay period.

Possible settings:

- **0** - disables the function.
- **Any positive integer** - The default is 100s. You may want a higher setting for large clusters.

For example, the client sets `Cluster_Synchronization_Delay` to 100s and sends a key creation request to appliance A, which is part of a cluster. Appliance A creates the key and automatically synchronizes with rest of the cluster. The client will use only appliance A for 100 seconds - enough time for the cluster synchronization to complete. After this time period, the client will use other cluster members as before.

CA_File

The `CA_File` parameter refers to the CA certificate that was used to sign the server certificate presented by the NAE Server to the client.

Possible settings:

- **The path and filename** - The path can be absolute or relative to your application. Don't use quotes, even if the path contains spaces.

Because all SafeNet KeySecures in a clustered environment must have an identical configuration, all servers in the cluster use the same server certificate. As such, you only need to point to one CA certificate in the `CA_File` system parameter. If you do not supply the CA certificate that was used to sign the server certificate used by the SafeNet KeySecures, your client applications cannot establish SSL connections with any of the servers in the cluster.

If a local CA on the SafeNet KeySecure was used to sign the NAE Server certificate, you can download the certificate for the local CA, and put that certificate on the client.

Cert_File

The `Cert_File` parameter stores the path and filename of the client certificate. This is only used when your SSL configuration requires clients to provider a client certificate to authenticate to the SafeNet KeySecures.

Possible settings:

- **The path and filename** - The path can be absolute or relative to your application. Don't use quotes, even if the path contains spaces. Client certificates *must* be PEM encoded.

If this value is set, the certificate and private key must be present, even if the SafeNet KeySecure is not configured to request a client certificate.

Key_File

The `Key_File` parameter refers to the private key associated with the client certificate specified in the Cert_File parameter.

Possible settings:

- **The path and filename** - The path can be absolute or relative to your application. Don't use quotes, even if the path contains spaces. The client private key must be in PEM-encoded PKCS#12 format.

Because this key is encrypted, you must use the `Passphrase` parameter so the SafeNet KeySecure can decrypt it.

IMPORTANT: If this value is set, the certificate and private key must be present, even if the SafeNet KeySecure is not configured to request a client certificate.

Passphrase

The `Passphrase` parameter refers to the passphrase associated with the private key.

Possible settings:

- The passphrase associated with the private key named in Key_File

If you do NOT provide this passphrase, the client attempts to read the passphrase from standard input; this causes the application to hang.

Remember that the properties file is NOT encrypted. Make sure that this file resides in a secure directory and has appropriate permissions so that it is readable only by the appropriate application or user.

Log_Level

The `Log_Level` parameter determines the level of logging performed by the client.

Possible settings:

- **NONE** – disables client logging. We recommend that you not disable logging.
- **LOW** – only error messages are logged.
- **MEDIUM** – the client logs error messages and warnings.
- **HIGH** – the client logs error messages, warnings and informational messages. This level generates a very large number of entries and is usually reserved for debugging.

IMPORTANT: The user running your client application must have permission to write to the log file, and to create new files in the directory where the log files are created.

Log_File

The `Log_File` parameter specifies a name (and possibly a path) for the log file.

Possible settings:

- **a filename** - The log will be created in the same directory as the client. The default value is Logfile.txt.
- **a path and filename** - The path can be absolute or relative to your application. Don't use quotes, even if the path contains spaces.

Log_Rotation

The `Log_Rotation` parameter specifies whether logs are rotated daily or once they reach a certain size.

Possible settings:

- **Daily** - Rotates logs daily. This is the default.
- **Size** - Rotates logs when the reach the size specified in Log_Size_Limit.

Log_Size_Limit

The `Log_Size_Limit` parameter specifies how large log files can be before they are rotated. This parameter is used only when `Log_Rotation` is set to **Size**.

Possible settings:

- **Any positive integer** - The default unit is bytes. You can use the suffix k (or K) for kilobytes and m (or M) for megabytes. The default value is 100k.

Setting up SSL

This chapter provides an overview of SafeNet's SSL and SSL with Client Certificate Authentication features, and provides a walkthrough of both configuration procedures.

SSL Overview

Standard SSL communication requires a certificate that identifies the server. This certificate is signed by a certificate authority (CA) known to both the server and the client. During the SSL handshake, the server certificate is passed to the client. The client uses a copy of the CA certificate to validate the server certificate, thus authenticating the server.

While the CA can be a third-party CA or your corporate CA, you will most likely use a local CA on the SafeNet KeySecure appliance. If you are not using a local CA, consult your CA documentation for instructions on signing requests and exporting certificates.

TIP: SafeNet recommends that you increase security only after confirming network connectivity. You should establish a TCP connection before enabling SSL. Otherwise, an unrelated network connection mistake could interfere with your SSL setup and complicate the troubleshooting process.

To use an SSL connection when communicating with the SafeNet KeySecure appliance, you must configure both the server and the client.

To configure the server, you must:

1. Create a server certificate. (If you're using a cluster, each member must have its own, unique certificate.)

   This may involve the following steps:

   a. Creating a Local CA.

   b. Creating a Server Certificate Request on the Management Console.

   c. Signing a Server Certificate Request with a Local CA.

   Update the Cryptographic Key Server settings on the Management Console (Device, Key Server, and Key Server). You'll need to check **Use SSL** and select your server certificate in the **Server Certificate** field.

To configure the client, you must:

2. Place a copy of the CA certificate on your client.

   This may involve Downloading the Local CA Certificate.

3. Update `IngrianNAE.properties` file as follows:

```
Protocol=ssl
CA_File=<location and name of the CA certificate file>
```

SSL Configuration Procedures

This section describes the procedures you will follow when configuring SSL. It explains the following processes:

Creating a Local CA

To create a local CA:

1. Log on to the Management Console as an administrator with Certificate Authorities access control.
2. Navigate to the **Create Local Certificate Authority** section on the Certificate and CA Configuration page (Security, CAs & SSL Certificates, Local CAs).
3. Modify the fields as needed.
4. Select either *Self-signed Root CA* or *Intermediate CA Request* as the **Certificate Authority Type**.
5. Click **Create**.

NOTE: Only a local CA can sign certificate requests on the SafeNet KeySecure appliance. If you are using a CA that does not reside on the SafeNet KeySecure appliance you cannot use the Management Console to sign certificate requests.

Creating a Server Certificate Request on the Management Console

To create a server certificate request on the Management Console:

1. Log on to the Management Console as an administrator with Certificates access control.
2. Navigate to the **Create Certificate Request** section of the Certificate Configuration page (Security, CAs & SSL Certificates, SSL Certificates) and modify the fields as needed.
3. Click **Create Certificate Request**. This creates the certificate request and places it in the Certificate List section of the Certificate and CA Configuration page. The new entry shows that the **Certificate Purpose** is *Certificate Request* and that the **Certificate Status** is *Request Pending*.

Signing a Server Certificate Request with a Local CA

To sign a server certificate request with a local CA:

1. Log in to the **Management Console** as an administrator with **Certificates and Certificate Authorities** access controls.
2. Navigate to the Certificate List section on the Certificate and CA Configuration page (Security, CAs & SSL Certificates, SSL Certificates).
3. Select the certificate request and click **Properties**.

4. Copy the text of the certificate request. The copied text must include the header **(-----BEGIN CERTIFICATE REQUEST-----)** and footer **(-----END CERTIFICATE REQUEST-----)**.
5. Navigate to the **Local Certificate Authority List** (Security, CAs & SSL Certificates, Local CAs). Select the local CA and click **Sign Request** to access the Sign Certificate Request section.
6. Modify the fields as shown:
   - **Sign with Certificate Authority** - Select the CA that signs the request.
   - **Certificate Purpose** - Select *Server*.
   - **Certificate Duration (days)** - Enter the life span of the certificate.
   - **Certificate Request** - Paste all text from the certificate request, including the header and footer.
7. Click **Sign Request**. This will take you to the CA Certificate Information section.
8. Copy the actual certificate. The copied text must include the header **(-----BEGIN CERTIFICATE-----)** and footer **(-----END CERTIFICATE-----)**.
9. Navigate back to the **Certificate List** section (Security, CAs & SSL Certificates, SSL Certificates). Select your certificate request and click **Properties**.
10. Click **Install Certificate**.
11. Paste the actual certificate in the **Certificate Response** text box. Click **Save**. The Management Console returns you to the Certificate List section. The section will now show that the **Certificate Purpose** is *Server* and that the **Certificate Status** is *Active*.

The certificate can now be used as the server certificate for the NAE Server.

Importing a Server Certificate to the SafeNet KeySecure Appliance

As an alternative to the certificate creation procedure outlined above, you can import a certificate to the SafeNet KeySecure appliance.

To import a certificate to the SafeNet KeySecure appliance:

1. Log in to the **Management Console** as an administrator with **Certificates** access control.
2. Navigate to the **Import Certificate** section of the **Certificate and CA Configuration** page (Security, CAs & SSL Certificates, SSL Certificates).
3. Select the method used to import the certificate file.
4. Enter the name of the file and the private key password.
5. Click the **Import Certificate** button.

The certificate can now be used as the server certificate for the NAE Server.

Downloading the Local CA Certificate

To download a local CA certificate from the SafeNet KeySecure appliance:

1. Log in to the **Management Console** as an administrator with **Certificate Authorities** access control.
2. Navigate to the **Local Certificate Authority List** section of the Certificates and CA Configuration page (Security, CAs & SSL Certificates, Local CAs).
3. Select the **Local CA** and click **Download** to download the file to your client. You should place the CA certificate in a secure location and modify access appropriately.

NOTE: Use the **CA_File** parameter in the **IngrianNAE.properties** file to indicate the name and location of the CA certificate.

SSL Walkthrough for SafeNet Clients

This walkthrough assumes the following:

- You have read the SSL overview section.
- You have configured a TCP connection between your client and the SafeNet KeySecure appliance.

There are a few different ways that you could configure SSL. For example, you can use a CA that does not reside on the SafeNet KeySecure appliance or you can create a new CA. This walkthrough makes such decisions for you. By following these instructions, you will:

1. Create a Local CA.
2. Create a Certificate Request.
3. Create a Server Certificate by signing the Certificate Request with the Local CA.
4. Download the Local CA to the client.

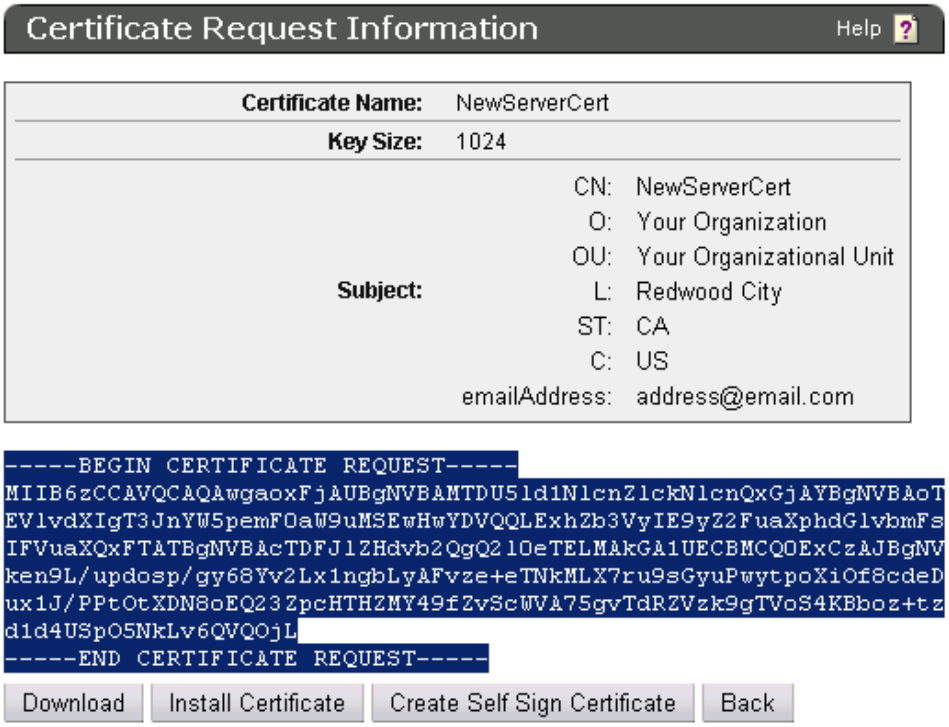
Once you have completed and understood this walkthrough, you might decide to alter some of the steps to better fit your organization's policies.

To configure SSL:

1. Log in to the **Management Console** as an administrator with Certificates, Certificate Authorities, and NAE Server access controls.
2. Navigate to the **Create Local Certificate Authority** section (Security, CAs & SSL Certificates, Local CAs). Enter the values shown below to create a new local CA. Click **Create**.

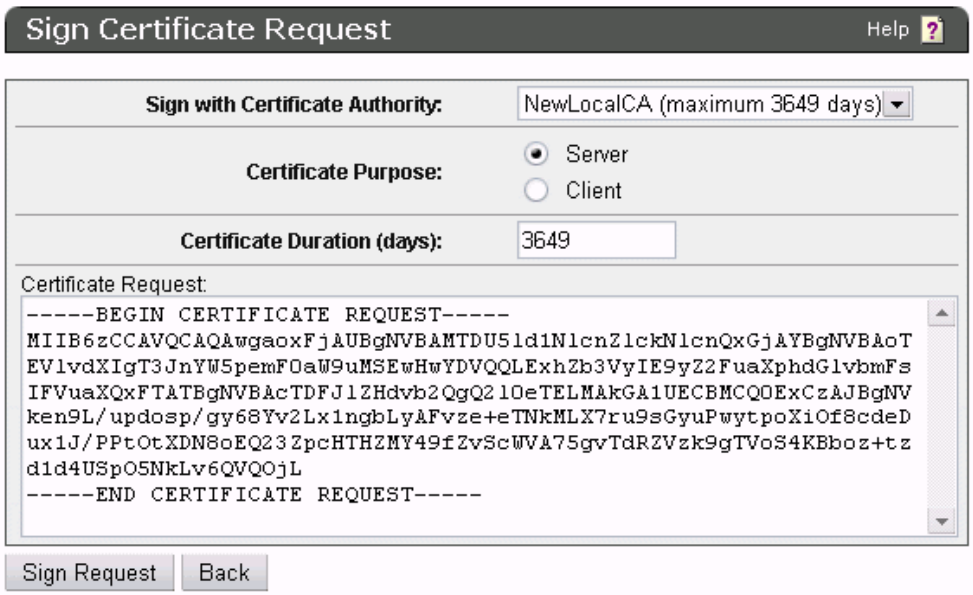
3. Navigate to the **Create Certificate Request** section (Security, CAs & SSL Certificates, SSL Certificates). Enter the values shown below to create a request. Click **Create Certificate Request**.

4. Select your new certificate request from the **Certificate List** section (right above the Create Certificate Request section). Click **Properties**. Copy the actual request (highlighted below). Include the header and footer.



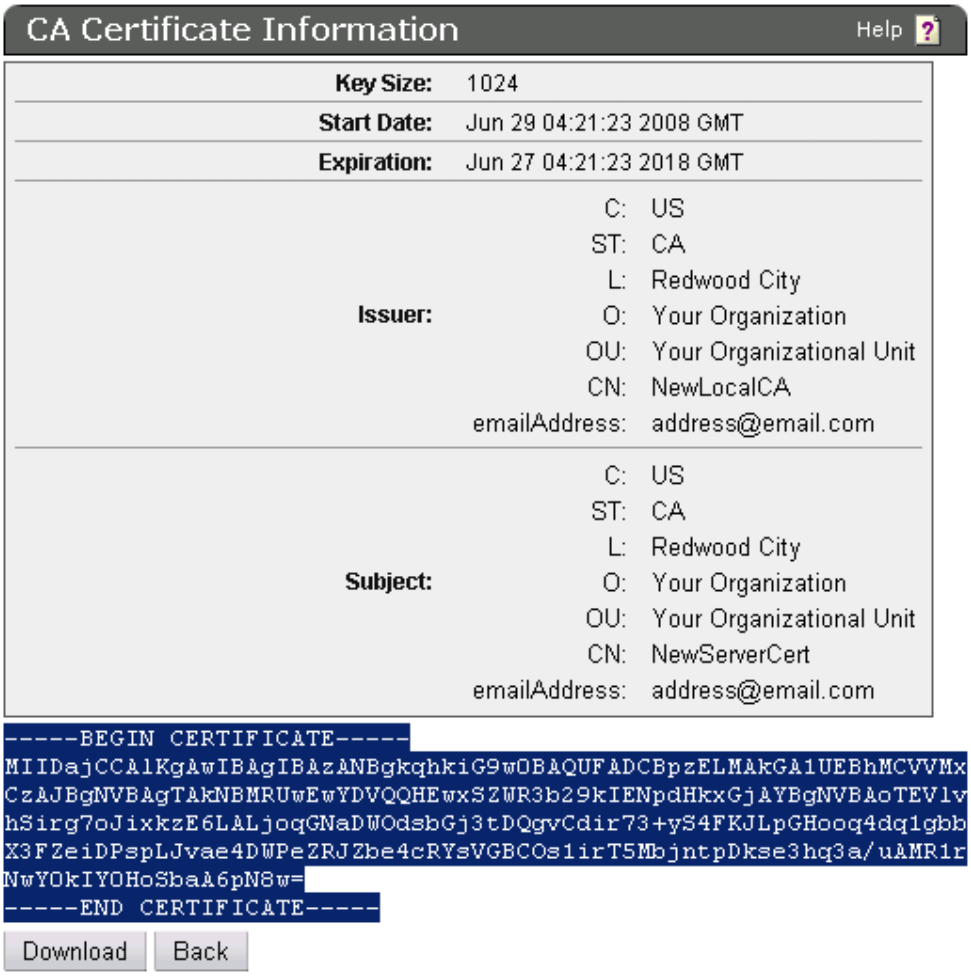
5. Navigate back to the **Local Certificate Authority List** section (Security, CAs & SSL Certificates, Local CAs). Select your new local CA and click **Sign Request**.

6. Select **Certificate Purpose** *Server* and paste the certificate request into the **Certificate Request** text box, as shown below.

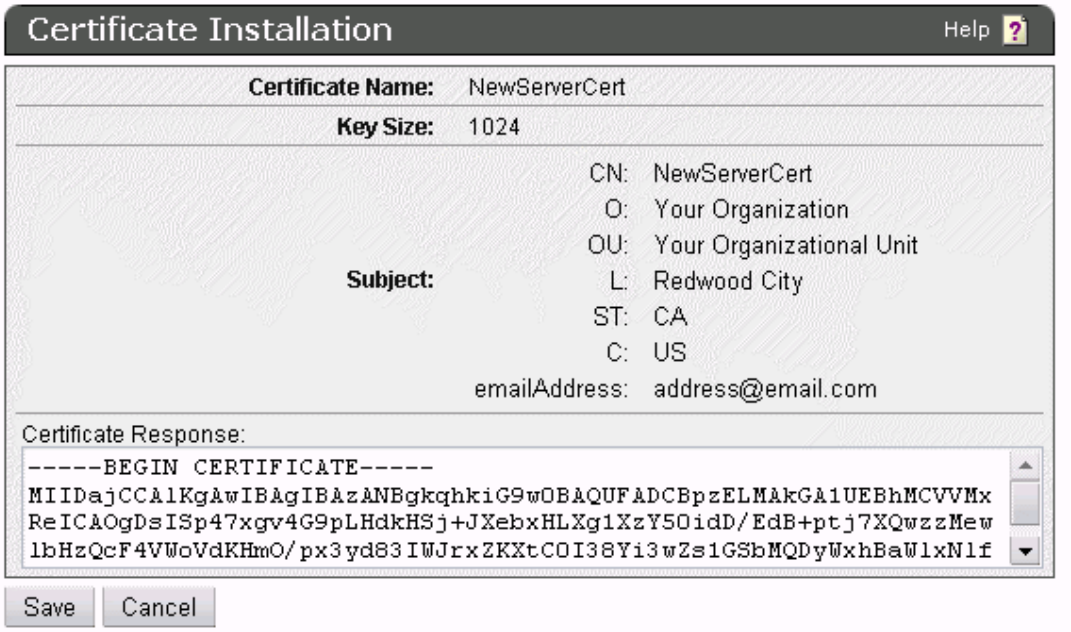


7. Click **Sign Request**. This will take you to the CA Certificate Information section.

8. Copy the actual certificate (highlighted below). Include the header and footer.



9. Navigate back to the **Certificate List** section (Security, CAs & SSL Certificates, SSL Certificates). Select your certificate request and click **Properties**.
10. Click **Install Certificate**.
11. Paste the actual certificate, as shown below. Click **Save**.



12. The **Certificate List** section will now indicate that **NewServerCert** is an active certificate.
13. Navigate to the **NAE Server Settings** section (Device, Key Server, Key Server). Click **Edit**.
14. Select **Use SSL** and select your new server certificate in the Server Certificate field. Click **Save**.
15. Navigate back to the **Local Certificate Authority List** section (Security, CAs & SSL Certificates, Local CAs). Select your new **CA** and click **Download**. Place the CA certificate in a secure directory on your client.
16. Update the following parameters in your IngrianNAE.properties file:
    - `Protocol=ssl`
    - `CA_File=<path to CA cert>\localca.crt`

> **NOTE:** Whenever you update the properties file, you must restart the database for the changes to take effect.

SSL with Client Certificate Authentication Overview

This SSL implementation requires that both the server and the client produce certificates. Each certificate is signed by a trusted CA known to both the server and the client. Most likely, you will use one CA to sign both certificates. During the SSL handshake, the certificates are exchanged. Both the client and the server use the CA certificate to validate one another's certificate, thus authenticating the other party.

> **NOTE:** For more information about setting up SSL, see SSL Overview on page 24.

To enable client certificate authentication, *you must first successfully configure SSL*. Then, you must make additional configuration changes to the client and server.

Document PN: 007-013430-001, Rev B, © Gemalto 2015 - 2017,

TIP: SafeNet recommends that you increase security *only after* confirming network connectivity. You should establish an SSL connection before enabling Client Certificate Authentication. Otherwise, an SSL configuration mistake could interfere with your Client Certificate Authentication setup and complicate the troubleshooting process.

To configure the client, you must:

1. Create a client certificate.

   This may involve the following steps:

   a. Generating a Client Certificate Request with OpenSSL .

   b. Signing a Certificate Request and Downloading the Certificate.

   You can create a certificate request using OpenSSL. You can then sign the request with the local CA on the SafeNet KeySecure appliance. Once signed, the certificate request becomes a valid certificate.

   If you are not using a local CA, consult your CA documentation for instructions on signing requests and exporting certificates.

2. Update `IngrianNAE.properties` file as follows:

   - `Cert_File=<location and name of the client certificate>`
   - `Key_File=<location and name of the client's key file>`
   - `Passphrase=<the passphrase used to unlock the client's key file>`

   NOTE: Whenever you update the properties file, you must restart the database for the changes to take effect.

To configure the server, you must:

3. Place a copy of the CA certificate on your server.

   This may involve the following steps:

   a. Installing a CA Certificate on the Server.

   b. Adding a CA to a Trusted CA List Profile.

4. Update the Authentication Settings section on the Management Console (Device, Key Server, Key Server).

   You'll need to select either *Used for SSL session only* or *Used for SSL session and username* in the **Client Certificate Authentication** field. The profile listed in the **Trusted CA List Profile** field must include the CA used to sign the client certificate. You can update the other fields as needed.

SSL with Client Certificate Authentication Procedures

This section describes the procedures you will follow when configuring SSL with Client Certificate Authentication. It explains the following processes:

- "Generating a Client Certificate Request with OpenSSL" on page 33
- "Signing a Certificate Request and Downloading the Certificate" on page 33
- "Installing a CA Certificate on the Server" on page 34

Document PN: 007-013430-001, Rev B, © Gemalto 2015 - 2017,

- “Adding a CA to a Trusted CA List Profile” on page 34

Generating a Client Certificate Request with OpenSSL

To generate a client certificate request:

1. Open a command prompt window.
2. If you are using OpenSSL, use the following command:

```
openssl req -out clientreq -newkey rsa:2048 -keyout clientkey
```

> **NOTE:** The certificate request and private key will both be created in the working directory by default. You can generate them in another directory by including a location in the request and key names.
>
> For example, to create them in the C:\client_certs folder, use the following command:
>
> **openssl req -out C:\client_certs\clientreq -newkey rsa:2048 -keyout C:\client_certs\clientkey**

The key generation process will then request the following data:

- A PEM passphrase to encode the private key.

  The passphrase that encodes the private key is the first passphrase you provide after issuing the command above. This will be the Passphrase parameter in the **IngrianNAE.properties** file.

- The distinguished name.

  The distinguished name is a series of fields whose values are incorporated into the certificate request. These fields include country name, state or province name, locality name, organization name, organizational unit name, common name, email address.

> **NOTE:** For more information about deriving NAE usernames and authenticating client IP addresses, see “Authentication Overview” in the KeySecure Appliance User Guide.

- A challenge password.

  This challenge password is NOT used in the SafeNet KeySecure environment.

- An optional company name.

Signing a Certificate Request and Downloading the Certificate

This section describes how to sign a certificate request with a local CA and then download the certificate. You must download the certificate *immediately* after it is signed by the CA.

To sign a certificate request with a local CA:

1. Open the certificate request in a text editor.
2. Copy the text of the certificate request. The copied text must include the header (-----BEGIN CERTIFICATE REQUEST-----) and the footer (-----END CERTIFICATE REQUEST-----).

3. Log in to the SafeNet KeySecure appliance as an administrator with Certificate Authorities access control.
4. Navigate to the Local Certificate Authority List (Security, CAs & SSL Certificates, Local CAs). Select the local CA and click **Sign Request** to access the Sign Certificate Request section.
5. Modify the fields as shown:
   - **Sign with Certificate Authority** - Select the CA that signs the request.
   - **Certificate Purpose** - Select **Client**.
   - **Certificate Duration (days)** - Enter the life span of the certificate.
   - **Certificate Request** - Paste all text from the certificate request, including the header and footer.
6. Click **Sign Request**. This will take you to the CA Certificate Information section.
7. Click the **Download** button to save the certificate on your local machine. You should place the certificate in a secure location and modify access appropriately.

NOTE: Use the **Cert_File** parameter in the **IngrianNAE.properties** file to indicate the name and location of the client certificate.

Installing a CA Certificate on the Server

If the client certificate was signed by a non-local CA, you must install the CA certificate on the SafeNet KeySecure appliance. To install a CA Certificate:

1. Log in to the SafeNet KeySecure appliance as an administrator with Certificate Authorities access control.
2. Navigate to **the Install CA Certificate** section on the Certificate and CA Configuration page (Security, CAs & SSL Certificates, Known CAs).
3. Enter the **Certificate Name**.
4. Paste all text from the certificate in the **Certificate** field, including the header and footer.
5. Click **Install**.

Adding a CA to a Trusted CA List Profile

To add the CA that signed the client certificate to the Trusted CA List Profile:

1. Log in to the SafeNet KeySecure appliance as an administrator with Certificate Authorities access control.
2. Navigate to the **Trusted Certificate Authority List Profiles** section on the Certificate and CA Configuration page (Security, CAs & SSL Certificates, Trusted CA Lists).
3. Select the profile to which you want to add the CA.
4. Click **Properties**.
5. Click **Edit** in the Trusted Certificate Authority List section.
6. Select the CA in the **Available CAs** field and click **Add**. This moves your CA from the Available CAs field to the Trusted CAs field.
7. Click **Save**.

NOTE: To enable SSL with Client Certificate Authority, the Profile containing the CA that signed the client certificate must be selected as the **Trusted CA List Profile** on the Authentication Settings section.

SSL with Client Certificate Authentication Walkthrough for SafeNet KeySecure Clients

This walkthrough assumes the following:

- You have read the SSL with Client Certificate Authentication overview section.
- You have successfully completed the SSL Walkthrough for SafeNet KeySecure Clients. *You must use the Local CA created in that walkthrough.* The instructions below assume that the client and server certificates were signed by the same local CA.

There are a few different ways that you could configure SSL with Client Certificate Authentication. For example, you can use a CA that does not reside on the SafeNet KeySecure appliance or you can create a new CA . This walkthrough makes such decisions for you. By following these instructions, you will:

- Create a Client Certificate Request using OpenSSL.
- Create a Client Certificate by signing the Client Certificate Request with the Local CA on the SafeNet SafeNet KeySecure appliance.
- Add the Local CA to the Trusted CA List.

Once you have completed and understood this walkthrough, you might decide to alter some of the steps to better fit your organization's policies.

To configure client certificate authentication:

1. Open a command prompt window on the client.
2. If you are using OpenSSL, use the following command:

   ```
   openssl req -out ssl\clientreq -newkey rsa:2048 -keyout ssl\clientkey
   ```

   The key generation process will then request the following data:

   - A PEM passphrase to encode the private key.

   The passphrase that encodes the private key is the first passphrase you provide after issuing the command above. This will be the Passphrase parameter in the **IngrianNAE.properties** file.

   - The distinguished name.

   The distinguished name is a series of fields whose values are incorporated into the certificate request. These fields include country name, state or province name, locality name, organization name, organizational unit name, common name, email address.

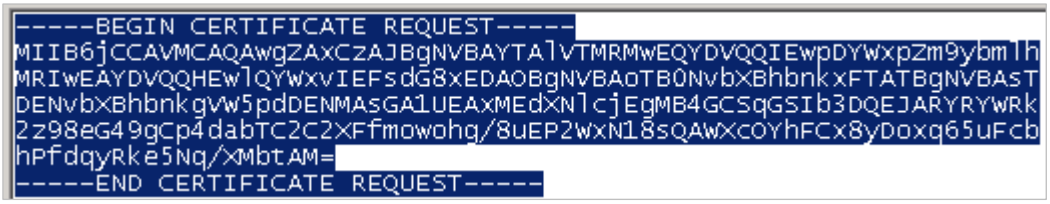
   NOTE: For more information about deriving NAE usernames and authenticating client IP addresses, see "Authentication Overview" in the KeySecure Appliance User Guide.

   - A challenge password.

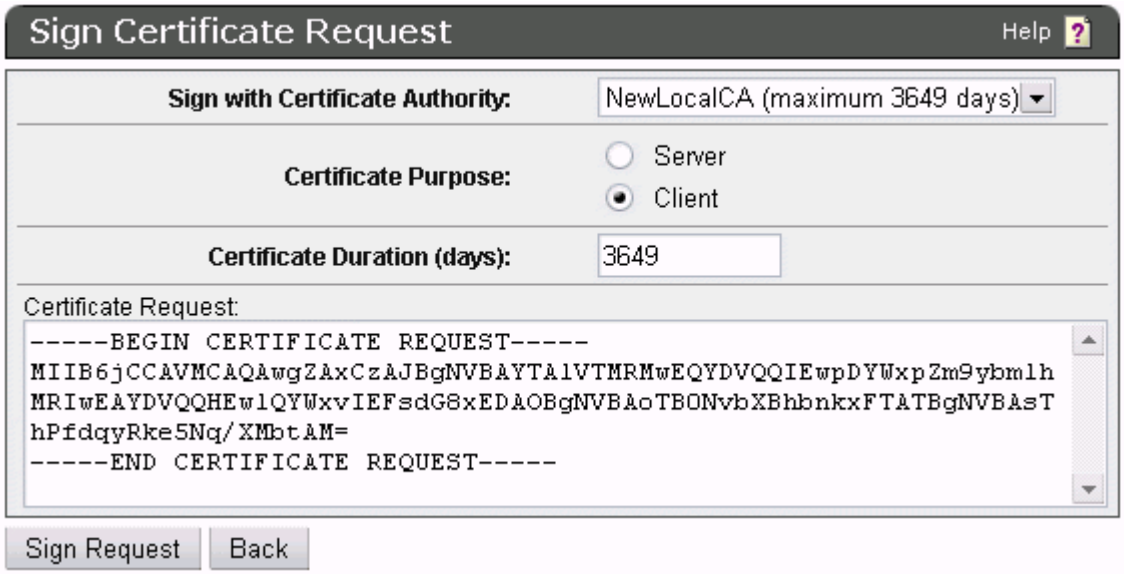
This challenge password is NOT used in the SafeNet KeySecure environment.

- An optional company name.

3. Open the client certificate request file and copy the actual request (highlighted below). Include the header and footer.



4. Log in to the Management Console as an administrator with Certificate Authorities access control.
5. Navigate to the Local Certificate Authority List section (Security, CAs & SSL Certificates, Local CAs). Select **NewLocalCA** and click **Sign Request**. (NewLocalCA is the CA you created in the SSL Walkthrough.)
6. Select **Certificate Purpose** *Client* and paste the certificate request into the **Certificate Request** field, as shown below.



7. Click **Sign Request**. This will take you to the CA Certificate Information section.
8. Click **Download** to download your new client certificate (`signed.crt`) to your client. Place the certificate in a secure directory on your client.
9. Navigate to the **Trusted Certificate Authority List Profiles** section (Security, CAs & SSL Certificates, Trusted CA Lists). Select **Default** as **Profile Name** and click **Properties**.
10. Click **Edit** in the Trusted Certificate Authority List section.
11. Select **NewLocalCA** in **Available CAs** and click **Add**. Click **Save**.
12. Update the following parameters in the IngrianNAE.properties file:

```
Cert_File=<path to client cert>\client.crt
Key_File=<path to client key>\clientkey
Passphrase=<the passphrase used to unlock the client's key file>
```

NOTE: Whenever you update the properties file, you must restart the database for the changes to take effect.

13. Return to the Management Console and navigate to the Authentication Settings section (Device, Key Server, Key Server) and enter the following values:
    - **Client Certificate Authentication**: Used for SSL Session only
    - **Trusted CA List Profile**: Default

IMPORTANT: The CA used to sign the client certificate must be a member of the **Trusted CA List Profile**.

6
Managing HSM Wallets

Encryption wallet can be of two types: HSM (SafeNet KeySecure) and Oracle (software) wallets. It is recommended to use an HSM rather than Oracle wallet. This chapter describes instructions to configure an HSM wallet.

NOTE: The steps described in this chapter assume that no HSM or Oracle wallet already exists. This chapter describes steps to perform fresh configuration of an HSM wallet. For information on configuring Oracle wallets, see 6, “Managing Oracle Wallets”.

This chapter contains the following information:

Configuring HSM Wallets

After configuring SafeNet PKCS#11 with Oracle TDE, the master key for TDE can be generated and stored on HSM. Generation of the master key involves the following steps:

- Adding the Wallet Location to sqlnet.ora
- Restarting the Oracle Database
- Creating the Encryption Wallet
- Encrypting a Column in a Table
- Closing the Wallet

Adding the Wallet Location to sqlnet.ora

On Linux/UNIX

Add the following code to the `$ORACLE_HOME/network/admin/sqlnet.ora` file.

```
ENCRYPTION_WALLET_LOCATION =
 (SOURCE = (METHOD = HSM)
  (METHOD_DATA = (DIRECTORY = <desired_path>))
 )
```

On Windows

Add the following to $ORACLE_HOME\NETWORK\ADMIN\sqlnet.ora.

```
ENCRYPTION_WALLET_LOCATION =
 (SOURCE =
  (METHOD = HSM)
  (METHOD_DATA = (DIRECTORY = %SYSTEM_DRIVE%\<desired_path>))
 )
```

NOTE:

- It is recommended to use an HSM rather than Oracle wallet. However, if an Oracle wallet is to be used, set (METHOD = **FILE**) instead of **HSM**. For details, see Chapter 6, “Managing Oracle Wallets”.
- In the above code, <desired_path> specifies the location where the wallet will be stored. Make sure that <desired_path> specified by the DIRECTORY parameter exists and the Oracle user has appropriate permissions on it. From this point onward, this document uses /home/oracle/WALLET as <desired_path> (the wallet location).
- For Oracle 11.2 and higher, the METHOD_DATA entry is not needed. The code change is as follows:

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

Restarting the Oracle Database

After making any changes to the sqlnet.ora file, it is necessary to restart the database for the changes to be effective. To restart the database, execute the following commands:

```
$ sqlplus /nolog
SQL> connect / as sysdba
Connected
SQL> shutdown immediate
Database closed.
Database dismounted.
ORACLE instance shut down.
SQL> startup
ORACLE instance started.
Total System Global Area 167772160 bytes
Fixed Size …… 1247900 bytes
Variable Size …… 75498852 bytes
```

```
Database Buffers …… 88080384 bytes
Redo Buffers …… 2945024 bytes
Database mounted.
Database opened.
SQL> exit
$ lsnrctl stop
$ lsnrctl start
```

Creating the Encryption Wallet

After the database is restarted, the encryption wallet can be created to store the master encryption keys.

In Oracle 11g R2, a unified master encryption key is created. This key can be used for both TDE column and TDE tablespace encryption. The master encryption key can be created, stored, and rotated in the HSM.

To create an encryption wallet (if it does not already exist at the location specified in the `sqlnet.ora` file.) and add a master encryption key to it, execute the following commands:

```
$ sqlplus system/<system_password>
SQL> alter system set encryption key identified by "tdeowner:asdf1234";
```

NOTE:

- In this sample command, `"tdeowner:asdf1234"` represents the NAE user name and its password. The NAE user name and password are case-sensitive. They must appear in double-quotes (") and be separated by a colon (:).
- The NAE user specified in the above command is the owner of the encryption key created and stored on SafeNet KeySecure.
- The Key and Policy Configuration page of SafeNet KeySecure Management Console displays the generated master encryption key.

TIP: Re-executing the command `“alter system set encryption key identified by "tdeowner:asdf1234";”` rotates (re-keys) the master encryption key. Every time this command is executed, a new encryption key is generated and stored on SafeNet KeySecure, and the data is encrypted with the new encryption key.

Encrypting a Column in a Table

After the master encryption key is generated, it can be used to encrypt a column of a table or tablespace. To encrypt a column in a table, execute the following command:

```
SQL> alter table <table_owner>.<table_name> modify (<column_name> encrypt);
```

Here <table_owner> represents the name of the table owner and <table_name> represents the name of the table containing the column <column_name> to be encrypted.

For example:

```
SQL> alter table appowner.credit_cards modify (ccnum encrypt);
```

To verify whether the encrypted data is accessible, execute the following command:

```
SQL> select * from <table_owner>.<table_name>;
```

For example:

```
SQL> select * from appowner.credit_cards;
```

NOTE: On Windows platforms, while performing select operations on the encrypted table, the following error may occur: `ORA-28353: failed to open wallet`. This error occurs because the wallet gets automatically closed after exiting a SQL session. Therefore, the wallet must be opened manually before performing any operations on the encrypted table.

Opening the Wallet Manually

To open the wallet, execute:

```
alter system set wallet open identified by "tdeowner:asdf1234";
```

Checking the Wallet Status

To check the wallet status, execute:

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

Oracle TDE/PKCS#11 can also be used to perform tablespace encryption.

Closing the Wallet

After encrypting a column in a table, close the wallet by executing:

```
SQL> alter system set wallet close identified by "tdeowner:asdf1234";
SQL> exit;
```

IMPORTANT: Oracle TDE/PKCS#11 can also be used to enable auto login wallets and to perform tablespace encryption. For details, see “Enabling Auto Login Wallet” below.

Enabling Auto Login Wallet

When an encrypted wallet is password-protected, then a valid password is needed to open the wallet. After the wallet is opened, the authorized users and applications can perform operations on the encrypted data.

To eliminate the need for supplying password, auto-open (auto login) wallet can be created. With auto-open wallet, the encrypted wallet is automatically opened as soon as the database is started. Authorized users and applications can access the encrypted data without supplying password.

NOTE: An auto-open wallet should be created from an existing encryption wallet. This ensures that the master key can be transferred to the new auto-open wallet.

WARNING: Do not delete the encryption wallet after creating an auto-open wallet. The encryption wallet is needed for rotation (or re-key) of the master encryption key.

Enabling Auto Login with HSM

These steps assume that Oracle TDE is used only in “HSM” mode, i.e., it is never migrated from an Oracle wallet.

To enable auto login with HSM:

1. Make sure that the wallet is open.

   To check the wallet status, execute:

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

   If the wallet is closed, open the wallet by executing:

   ```
   alter system set wallet open identified by "tdeowner:asdf1234";
   ```

2. Modify the existing **sqlnet.ora** file. Add the following code, if needed:

   ```
   (METHOD_DATA = (DIRECTORY =/home/oracle/WALLET)))
   ```

   Make sure that the path specified by the `DIRECTORY` parameter exists and the Oracle user has appropriate permissions on it.

   The content in the updated **sqlnet.ora** file should look similar to the following:

   ```
   ENCRYPTION_WALLET_LOCATION =
   (SOURCE = (METHOD = HSM)
   (METHOD_DATA = (DIRECTORY =/home/oracle/WALLET)))
   ```

3. Navigate to `/home/oracle/WALLET` directory.
4. Create a (local) auto-open encryption wallet at `/home/oracle/WALLET`. Execute the following command:

   ```
   $ orapki wallet create -wallet . -auto_login
   ```

   Here `.` represents the current directory, which should be `/home/oracle/WALLET`. Enter this path if the current directory is not `/home/oracle/WALLET`.

   Enter and confirm a password for the wallet when prompted.

   This directory now contains two files, **cwallet.sso** and **ewallet.p12**.

5. Enable auto-open HSM. Add the following entry to the empty wallet by executing:

   ```
   $ mkstore -wrl . -createEntry ORACLE.TDE.HSM.AUTOLOGIN any-non-empty-string
   ```

Enter the wallet password (created in the previous step) when prompted.

6. Close the wallet (connection to the HSM). Execute the following command:

```
SQL> alter system set encryption wallet close identified by "tdeowner:asdf1234";
```

> **NOTE:** If the above command results in the error "`ORA-28365: wallet is not open`", it means the wallet is already closed.

7. Open the wallet again. Run the following command:

```
SQL> alter system set encryption wallet open identified by "tdeowner:asdf1234";
```

8. Validate auto login.

   Restart the database, and query a table in an encrypted tablespace. If the query succeeds, then auto-open has been enabled (because the wallet was automatically opened by Oracle without supplying the password).

From now onward, no password is required to access the data encrypted with the master key.

Encrypting Tablespaces

Oracle TDE tablespace allows encryption of an entire tablespace. However, an existing tablespace can't be encrypted. A new tablespace is created, which can be encrypted using Oracle TDE. All objects created in the encrypted tablespace are automatically encrypted. Tablespace encryption is useful for securing sensitive data in tables. Tablespace encryption eliminates the need for analyzing columns to determine which columns need to be encrypted.

Encrypting a Tablespace with Default Algorithm

To encrypt a tablespace with the default algorithm (AES128), execute the following commands:

```
CREATE TABLESPACE <encrypted_tablespace_name>
DATAFILE '<path>/<tablespace_file_name>.dbf' SIZE 128K
AUTOEXTEND ON NEXT 64K
ENCRYPTION
DEFAULT STORAGE(ENCRYPT);
```

Where, `<encrypted_tablespace_name>` is the name for the encrypted tablespace and "`<path>/<tablespace_file_name>.dbf`" is the path and name for the tablespace file that will be created.

For example:

```
CREATE TABLESPACE default_algo_tablespace
DATAFILE '/home/oracle/default_algo_tablespace_file.dbf' SIZE 128K
AUTOEXTEND ON NEXT 64K
ENCRYPTION
DEFAULT STORAGE(ENCRYPT);
```

Encrypting a Tablespace with Specific Algorithm

To encrypt a tablespace with a specific algorithm, execute the following commands:

```
CREATE TABLESPACE <encrypted_tablespace_name>
DATAFILE '<path>/<tablespace_file_name>.dbf' SIZE 128K
AUTOEXTEND ON NEXT 64K
ENCRYPTION USING '<algorithm_name>'
DEFAULT STORAGE(ENCRYPT);
```

Where, `<encrypted_tablespace_name>` is the name for the encrypted tablespace; **"**`<path>/<tablespace_file_name>.dbf`**"** is the path and name for the tablespace file that will be created; and `<algorithm_name>` is the name of the algorithm to be used for tablespace encryption.

For example:

```
CREATE TABLESPACE specific_algo_tablespace
DATAFILE '/home/oracle/specific_algo_tablespace_file.dbf' SIZE 128K
AUTOEXTEND ON NEXT 64K
ENCRYPTION USING 'AES256'
DEFAULT STORAGE(ENCRYPT);
```

7
Managing Oracle Wallets

Encryption wallet can be of two types: HSM (SafeNet KeySecure) and Oracle (software) wallets. This chapter describes instructions to configure an Oracle wallet. The chapter contains detailed steps to enable auto login with an Oracle wallet and steps to migrate keys from an existing Oracle wallet to HSM.

NOTE: The steps described in this chapter assume that no HSM or Oracle wallet already exists. For information on configuring HSM wallets, see Chapter 5, “Managing HSM Wallets”.

This chapter contains the following information:

Configuring an Oracle Wallet

After configuring SafeNet PKCS#11 with Oracle TDE, the encryption key for TDE can be generated and stored in the Oracle wallet (i.e., in database). Generation of the encryption key involves the following steps:

- Adding the Wallet Location to sqlnet.ora
- Restarting the Oracle Database
- Creating the Encryption Wallet
- Encrypting a Column in a Table
- Closing the Wallet

Adding the Wallet Location to sqlnet.ora

On Linux/UNIX

Add the following code to the `$ORACLE_HOME/network/admin/sqlnet.ora` file.

```
ENCRYPTION_WALLET_LOCATION =
 (SOURCE =
  (METHOD = FILE)
  (METHOD_DATA = (DIRECTORY = <desired_path>)
```

```
)
```

On Windows

Add the following to `$ORACLE_HOME\NETWORK\ADMIN\sqlnet.ora`.

```
ENCRYPTION_WALLET_LOCATION =
 (SOURCE =
  (METHOD = FILE)
  (METHOD_DATA = (DIRECTORY = %SYSTEM_DRIVE%\<desired_path>))
 )
```

NOTE:

- If an **HSM** wallet is to be used, set `(METHOD = HSM)` instead of FILE. For details, see Chapter 5, “Managing HSM Wallets”.
- In the above code, `<desired_path>` specifies the location where the wallet will be stored. Make sure that `<desired_path>` specified by the `DIRECTORY` parameter exists and the Oracle user has appropriate permissions on it. From this point onward, this document uses `/home/oracle/WALLET` as `<desired_path>` (the wallet location).
- For Oracle 11.2 and higher, the `METHOD_DATA` entry is not needed. The code change is as follows:

```
ENCRYPTION_WALLET_LOCATION =
 (SOURCE = (METHOD = FILE))
```

Restarting the Oracle Database

After making any changes to the `sqlnet.ora` file, it is necessary to restart the database for the changes to be effective. To restart the database, execute the following commands:

```
$ sqlplus /nolog
SQL> connect / as sysdba
Connected
SQL> shutdown immediate
Database closed.
Database dismounted.
ORACLE instance shut down.
SQL> startup
ORACLE instance started.
Total System Global Area 167772160 bytes
Fixed Size …… 1247900 bytes
```

```
Variable Size ……. 75498852 bytes
Database Buffers ……. 88080384 bytes
Redo Buffers ……. 2945024 bytes
Database mounted.
Database opened.
SQL> exit
$ lsnrctl stop
$ lsnrctl start
```

Creating the Encryption Wallet

After restarting the database, the encryption wallet can be created to store the encryption keys.

In Oracle 11g R2, a unified master encryption key is created. This key can be used for both TDE column and TDE tablespace encryption. The master encryption key can be created, stored, and rotated in the Oracle wallet.

To create an encryption wallet (at the location specified in the sqlnet.ora file) and add a master encryption key to it, execute the following commands:

1. `$ sqlplus system/<system_password>`

   At this point, no Oracle wallet exists. This can be confirmed by executing: `select * from v$encryption_wallet;`

   The command output will display `WRL_TYPE` (wallet type) as `file` and `STATUS` as `CLOSED`.

2. `SQL> alter system set encryption key identified by "asdf1234";`

NOTE:

- In this sample command, `"asdf1234"` represents the password to access the Oracle wallet. The password is case-sensitive and must appear in double-quotes (**"**).
- As the encryption key is stored in Oracle wallet, the command requires only the password, not the NAE user name. However, in case of HSM wallet, where the encryption key is created and stored on HSM (SafeNet KeySecure), the command requires both the NAE user name (the key owner) and its password.
- The above command creates the **ewallet.p12** file at `/home/oracle/WALLET`.

3. Confirm the wallet status. Execute the following command:

   `select * from v$encryption_wallet;`

4. The command output will display `WRL_TYPE` (wallet type) as `file` and `STATUS` as `OPEN`. This confirms that the Oracle wallet has been created.

TIP: Re-executing the command “`alter system set encryption key identified by "asdf1234";`” rotates (re-keys) the master encryption key.

Every time this command is executed, a new encryption key is generated and stored in Oracle wallet, and the data is encrypted with the new encryption key.

Encrypting a Column in a Table

To encrypt a column in a table, execute the following command:

```
SQL> alter table <table_owner>.<table_name> modify (<column_name> encrypt);
```

Here `<table_owner>` represents the name of the table owner and `<table_name>` represents the name of the table containing the column `<column_name>` to be encrypted.

For example:

```
SQL> alter table appowner.credit_cards modify (ccnum encrypt);
```

To verify whether the encrypted data is accessible, execute the following command:

```
SQL> select * from <table_owner>.<table_name>;
```

For example:

```
SQL> select * from appowner.credit_cards;
```

NOTE: On Windows platforms, while performing select operations on the encrypted table, the following error may occur: `ORA-28353: failed to open wallet`. This error occurs because the wallet gets automatically closed after exiting a SQL session. Therefore, the wallet must be opened manually before performing any operations on the encrypted table.

Opening the Wallet Manually

To open the wallet, execute:

```
alter system set wallet open identified by "asdf1234";
```

Checking the Wallet Status

To check the wallet status, execute:

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

Closing the Wallet

After encrypting a column in a table, close the wallet by executing:

```
SQL> alter system set wallet close identified by "asdf1234";
SQL> exit;
```

Enabling Auto Login Wallet

When an encrypted wallet is password-protected, then a valid password is needed to open the wallet. After the wallet is opened, the authorized users and applications can perform operations on the encrypted data.

To eliminate the need for supplying password, auto-open (Auto Login) wallet can be created. With auto-open wallet, the encrypted wallet is automatically opened as soon as the database is started. Authorized users and applications can access the encrypted data without supplying password.

NOTE: An auto-open wallet should be created from an existing encryption wallet. This ensures that the master key can be transferred to the new auto-open wallet.

WARNING: Do not delete the encryption wallet after creating an auto-open wallet. The encryption wallet is needed for rotation (or re-key) of the master encryption key.

Enabling Auto Login with Oracle Wallet

These steps assume that Oracle TDE is used only in **"FILE"** mode.

To enable auto login with Oracle wallet:

1. Make sure that the wallet is open.

   To check the wallet status, execute:

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

   If the wallet is closed, open the wallet by executing:

   ```
   alter system set wallet open identified by "asdf1234";
   ```

2. Modify the existing **sqlnet.ora** file. Add the following code, if needed:

   ```
   (METHOD_DATA = (DIRECTORY =/home/oracle/WALLET)))
   ```

   Make sure that the path specified by the `DIRECTORY` parameter exists and the Oracle user has appropriate permissions on it.

   The content in the updated **sqlnet.ora** file should look similar to the following:

   ```
   ENCRYPTION_WALLET_LOCATION =
   (SOURCE = (METHOD = FILE)
   (METHOD_DATA = (DIRECTORY =/home/oracle/WALLET)))
   ```

3. Restart the database to make any changes in the **sqlnet.ora** file to be effective. This step is required only if the **sqlnet.ora** file is modified.
4. Navigate to `/home/oracle/WALLET` directory.
5. Create a (local) auto-open encryption wallet at `/home/oracle/WALLET`. Execute the following command:

   ```
   $ orapki wallet create -wallet . -auto_login
   ```

Here `.` represents the current directory, which should be `/home/oracle/WALLET`. Enter this path if the current directory is not `/home/oracle/WALLET`.

Enter and confirm a password `(asdf1234)` for the wallet when prompted.

A new file, **cwallet.sso**, is created at the `/home/oracle/WALLET` directory. This directory now contains two files, **cwallet.sso** and **ewallet.p12**.

6. Close the wallet (connection to the HSM). Execute the following command:

```
SQL> alter system set encryption wallet close identified by "asdf1234";
```

7. Open the wallet again. Run the following command:

```
SQL> alter system set encryption wallet open identified by "asdf1234";
```

8. Validate auto login.

   Restart the database, and query a table in an encrypted tablespace. If the query succeeds, then auto-open has been enabled (because the wallet was automatically opened by Oracle without supplying the password).

From now onward, no password is required to access the data encrypted with the master key.

Migrating Oracle Wallet to HSM

An existing Oracle wallet can be migrated to HSM. After the wallet is migrated, auto login can also be enabled with the wallet migrated to HSM.

Migrating an Oracle wallet to HSM and enabling auto login with the migrated wallet involves the following steps:

1. **Migrating an Oracle Wallet to HSM**
2. **Configuring Auto Login with Migrated Wallet**

Migrating an Oracle Wallet to HSM

The steps described below assume that an auto login Oracle wallet is already created as described in “Configuring an Oracle Wallet” on page 45 and “Enabling Auto Login Wallet” on page 49.

To migrate an Oracle wallet to HSM:

1. Modify the **sqlnet.ora** file, as follows:

   Change `(SOURCE = (METHOD = FILE)` to `(SOURCE = (METHOD = HSM)`.

   The content in the updated **sqlnet.ora** file should look similar to the following:

```
ENCRYPTION_WALLET_LOCATION =
(SOURCE = (METHOD = HSM)
(METHOD_DATA = (DIRECTORY =/home/oracle/WALLET)))
```

2. Restart the database.
3. Execute the following command:

```
SQL> alter system set encryption key identified by "tdeowner:asdf1234" migrate
using "asdf1234";
```

   Here `"tdeowner:asdf1234"` represents the NAE user name and its password and `"asdf1234"` represents the password to access the existing Oracle wallet.

The Oracle wallet is now successfully migrated to HSM and a master encryption key is generated on SafeNet KeySecure.

4. Check the wallet status by executing:

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

The command output will display entries for `file` and `HSM` wallets each with the `STATUS` as `OPEN`.

5. Verify whether the data encrypted with Oracle wallet is accessible over HSM. Execute the following command:

```
SQL> select * from <table_owner>.<table_name>;
```

For example:

```
SQL> select * from appowner.credit_cards;
```

Once the encryption key is migrated to SafeNet KeySecure, encryption requests similar to the following appear in the activity logs on SafeNet KeySecure every three seconds.

```
[2012-08-07 17:12:01] INFO 192.168.1.22 [-] tdeowner 100003 Crypto
ORACLE.TDE.HSM.MK.06AF95FC1B8BFD4FA0BFFB6D9D68B36AE3 [op#1 ENCRYPT
AES/CBC/PKCS5Padding] - [Success] [-]
```

These encryptions are a result of Oracle's heartbeat functionality related to TDE, specifically, related to external HSMs.

NOTE: After migration of an auto login Oracle wallet to HSM, the auto login functionality is no longer available with the migrated wallet. Auto login needs to be reconfigured with the migrated wallet.

For instructions, see the next topic.

Configuring Auto Login with Migrated Wallet

To configure auto login with wallet migrated to HSM (SafeNet KeySecure):

1. Make sure that the wallet location in the **sqlnet.ora** file is:

```
ENCRYPTION_WALLET_LOCATION =
(SOURCE = (METHOD = HSM)
(METHOD_DATA = (DIRECTORY =/home/oracle/WALLET)))
```

Restart the database, if needed.

2. Navigate to `/home/oracle/WALLET` directory.
3. Enable auto-open HSM. Add the following entry to the empty wallet by executing:

```
$ mkstore -wrl . -createEntry ORACLE.TDE.HSM.AUTOLOGIN any-non-empty-string
```

Enter the wallet password (`asdf1234`) when prompted.

4. Close the wallet (connection to the HSM). Execute the following command:

```
SQL> alter system set encryption wallet close identified by "tdeowner:asdf1234";
```

NOTE: If the above command results in the error "ORA-28365: wallet is not

open", it means the wallet is already closed.

5. Open the wallet again. Run the following command:

```
SQL> alter system set encryption wallet open identified by "tdeowner:asdf1234";
```

6. Validate auto login.

   Restart the database, and query a table in an encrypted tablespace. If the query succeeds, then auto-open has been enabled (because the wallet was automatically opened by Oracle).

From now onward, no password is required to access the encrypted data with the master encryption key.

8

Integrating TDE with SafeNet KeySecure on Oracle 11gR2 RAC (11.2.0.3)

This chapter contains the following information:

Setting up SafeNet KeySecure for Transparent Data Encryption (TDE) with Oracle RAC

Managing Multiple Databases on RAC

Setting up SafeNet KeySecure for Transparent Data Encryption (TDE) 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. There are several ways to check the status of the RAC. The *srvctl* utility shows the current configuration and status of the RAC database.

```
$ . oraenv
ORACLE_SID = [oracle] ? orcl
The Oracle base has been set to /u01/app/oracle

$ srvctl config database -d orcl
Database unique name: orcl
Database name: orcl
Oracle home: /u01/app/oracle/product/11.2.0/dbhome_1
Oracle user: oracle
Spfile: +DATA/orcl/spfileorcl.ora
Domain: localdomain
Start options: open
Stop options: immediate
Database role: PRIMARY
Management policy: AUTOMATIC
Server pools: orcl
```

```
Database instances: orcl1,orcl2
Disk Groups: DATA
Mount point paths:
Services:
Type: RAC
Database is administrator managed

$ srvctl status database -d orcl
Instance orcl1 is running on node rac1
Instance orcl2 is running on node rac2
$
```

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

```
$ export ORACLE_SID=orcl1
[oracle@rac1 Desktop]$ sqlplus / as sysdba
SELECT inst_name FROM v$active_instances;

INST_NAME
---------------------------------------------------------------------------
rac1.localdomain:orcl1
rac2.localdomain:orcl2

exit
$
```

NOTE: The above example shows 2x1 architecture, where 2 is the number of nodes (rac1 and rac2) and 1 is the number of databases (orcl).

However; our library supports more complex architecture where, the number of nodes and databases may be higher, for example 3x3 i.e. 3 nodes (rac1, rac2 and rac3) and 3 databases (orclA, orclB and orclC).

Configuring the PKCS11 Provider on Oracle RAC Instances

Configuration of SafeNet PKCS#11 with Oracle TDE on RAC involves the following steps:

1. **Installing the Application Server License on SafeNet KeySecure**
2. **Creating Oracle TDE Authentication Credentials**
3. **Installing SafeNet PKCS#11 Library on Linux (to be performed on both RAC1 and RAC2)**
4. **Configuring Connection to SafeNet KeySecure (to be performed on both RAC1 and RAC2)**

Installing the Application Server License on SafeNet KeySecure

To install the Application Server License on SafeNet KeySecure:

1. Obtain an Application Server License file from SafeNet.
2. Install the license file on the SafeNet KeySecure.

   To install the license file:

   a. Log on to the **Management Console** as an administrator.

   b. Navigate to the System Information page (**Device >> System Information & Upgrade**).

   c. Select the method of installation and click **Upgrade/Install**. The machine will reboot.

Creating Oracle TDE Authentication Credentials

The Oracle database needs a user to authenticate to the SafeNet KeySecure. This user will own the keys generated by Oracle TDE.

To create Oracle TDE authentication credentials on SafeNet KeySecure:

1. Log on to the Management Console as an administrator with Users and Groups access control. (The `admin` account created during the SafeNet KeySecure installation has this access control.)
2. Navigate to the Local Users section of the User & Group Configuration page (**Security >> Local Authentication >> Local Users & Groups**).
3. Click **Add**.
4. Enter a user name in the **Username** field and password in the **Password** field. (This document assumes **tdeowner** as the user name and **asdf1234** as password.
5. Select the **User Administration Permission** and **Change Password Permission** check boxes.
6. Click **Save**.

Installing SafeNet PKCS#11 Library on Linux (to be performed on both RAC1 and RAC2)

To install SafeNet PKCS#11 library:

1. Download SafeNet PKCS#11 from SafeNet customer support site: https://serviceportal.safenet-inc.com. The software adheres to the following naming convention:

   ```
   SafeNet Part Number - Product Name - Product Version - File Format
   ```

   For example:

   ```
   610-013046-001_pkcs11_tde_linux_64b_v8.3.0.000-0xx.tar.gz
   ```

2. Log on to the Linux/UNIX client as Oracle user.
3. Extract the file using any standard archive utility.

   For example, execute:

   ```
   tar -xzf <source_directory/tar_file_name> -C <destination_directory>
   ```

4. Create the `/opt/oracle/extapi/<ARCH>/hsm/safenet/<VERSION>` directory. The Oracle user must have appropriate access permissions on /opt/.

Where <ARCH> is the system architecture (either 32 or 64), and <VERSION> is the SafeNet software version number (e.g., 8.3.0).

From this point onward, this document uses <ARCH> as **64** and <VERSION> as 8.3.0. If the system architecture and version is different, adjust these values accordingly.

5. Copy the library file `libIngPKCS11.so-8.3.0.000` from extracted `/SafeNet/PKCS11/lib` directory to `/opt/oracle/extapi/64/hsm/safenet/8.3.0`.

   For example:

   ```
   $ cp libIngPKCS11.so-8.3.0.000 /opt/oracle/extapi/64/hsm/safenet/8.3.0
   ```

   NOTE: The receiving directory is a fixed location. Oracle searches this directory. It cannot be changed. Changing the directory name results in a "`cannot find PKCS11 library`" error.

6. Copy the properties file `IngrianNAE.properties` from extracted `/SafeNet/PKCS11` directory to `/opt/oracle/extapi/64/hsm/safenet/8.3.0`.

   For example:

   ```
   $ cp IngrianNAE.properties /opt/oracle/extapi/64/hsm/safenet/8.3.0
   ```

7. Rename **libIngPKCS11.so-8.3.0.000** as **libIngPKCS11.so**.

   For example:

   ```
   $ mv libIngPKCS11.so-8.3.0.000 libIngPKCS11.so
   ```

Configuring Connection to SafeNet KeySecure (to be performed on both RAC1 and RAC2)

To configure connection to SafeNet KeySecure:

1. Configure the connection to SafeNet KeySecure.

   Enter the following values in the IngrianNAE.properties file (placed at `/opt/oracle/extapi/64/hsm/safenet/8.3.0`).

   - **NAE_IP** – IP address of the SafeNet KeySecure.
   - **NAE_Port** – 9000 (This is the default value).
   - **Log_Level** – MEDIUM (This is the default, but it can be set to HIGH for troubleshooting).
   - **Log_File** – Full path and file name. The Oracle user must have write permissions for the path and file. A public location such as `/tmp` is recommended.

2. Modify environment variables for the Oracle user.

   Make sure that the following environment variables are exported so that they are inherited by new Oracle server processes. Edit the shell profile (in many shells, the file is called **.profile** and is located in the home directory of the Oracle user.)

   ```
   export SFNT_HSMAPI_BASE=/opt/oracle/extapi/<32|64>/hsm/safenet/8.3.0
   export NAE_Properties_Conf_Filename=$SFNT_HSMAPI_BASE/IngrianNAE.properties
   export IngrianNAE_Properties_Conf_Slot_ID_Max=100
   export IngrianNAE_Properties_Conf_SessionID_Max=100
   ```

and run the following commands:

```
srvctl setenv database –d orcl –T
"NAE_Properties_Conf_Filename=/opt/oracle/extapi/64/hsm/safenet/8.3.0/IngrianNAE.properties"

srvctl setenv database –d orcl –T "IngrianNAE_Properties_Conf_SessionID_Max=100"

srvctl setenv database –d orcl -T "IngrianNAE_Properties_Conf_Slot_ID_Max=100"
```

where,

`orcl` is the name of the database.

3. Add the environment variable (AIX and Solaris).
   - **AIX** – Add the environment variable, LIBPATH, as follows:

     ```
     export LIBPATH=/opt/oracle/extapi/64/hsm/safenet/8.3.0:
     /home/oracle/SafeNet/PKCS11/samplelibs
     ```

   - **Solaris** – Add the environment variable, LD_LIBRARY_PATH, as follows:

     ```
     export LD_LIBRARY_PATH=/opt/oracle/extapi/64/hsm/safenet/8.3.0:
     /home/oracle/SafeNet/PKCS11/samplelibs
     ```

Here, samplelibs is provided in the SafeNet PKCS#11 TDE package.

Managing Oracle Wallets

Encryption wallet can be of two types: HSM (SafeNet KeySecure) and Oracle (software) wallets. This section describes instructions to configure an Oracle wallet.

Configuring an Oracle Wallet

After configuring SafeNet PKCS#11 with Oracle TDE, the encryption key for TDE can be generated and stored in the Oracle wallet (i.e., in database). Generation of the encryption key involves the following steps:

1. **Adding the Wallet Location to sqlnet.ora**
2. **Restarting the Oracle Database**
3. **Creating the Encryption Wallet**
4. **Encrypting a Column in a Table**
5. **Opening the Wallet Manually**
6. **Checking the Wallet Status**
7. **Closing the Wallet**

Adding the Wallet Location to sqlnet.ora

a. 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 use the shared disk for storing the wallet. If shared disk is used you will not have to copy 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.

b. Create or add the following to `$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)))
```

Restarting the Oracle Database

After making any changes to the **sqlnet.ora** file, it is necessary to restart the database for the changes to be effective. To restart the database, execute the following commands:

```
$ sqlplus / as sysdba
SQL> connect / as sysdba
Connected
SQL> shutdown immediate
Database closed.
Database dismounted.
ORACLE instance shut down.
SQL> startup
ORACLE instance started.
Total System Global Area 167772160 bytes
Fixed Size 1247900 bytes
Variable Size 75498852 bytes
Database Buffers 88080384 bytes
Redo Buffers 2945024 bytes
Database mounted.
Database opened.
SQL> exit
$ lsnrctl stop
$ lsnrctl start
```

Creating the Encryption Wallet

After restarting the database, the encryption wallet can be created to store the encryption keys.

In Oracle 11g R2, a unified master encryption key is created. This key can be used for both TDE column and TDE tablespace encryption. The master encryption key can be created, stored, and rotated in the Oracle wallet.

To create an encryption wallet (at the location specified in the sqlnet.ora file) and add a master encryption key to it, execute the following commands:

1. `$ sqlplus system/<system_password>`

   At this point, no Oracle wallet exists. This can be confirmed by executing:

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

   The command output will display `WRL_TYPE` (wallet type) as `file` and `STATUS` as `CLOSED`.

2. `SQL> alter system set encryption key identified by "asdf1234";`

   Copy the **ewallet.p12** file created in the directory `/etc/oracle/wallet/RAC` from RAC1 to RAC2 in the same directory on RAC2.

   NOTE:

   - In this sample command, "`asdf1234`" represents the password to access the Oracle wallet. The password is case-sensitive and must appear in double-quotes (**"**).
   - As the encryption key is stored in Oracle wallet, the command requires only the password, not the NAE user name. However, in case of HSM wallet, where the encryption key is created and stored on HSM (SafeNet KeySecure), the command requires both the NAE user name (the key owner) and its password.
   - The above command creates the ewallet.p12 file at `/etc/oracle/wallet/RAC`.

3. Confirm the wallet status. Execute the following command:

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

   The command output will display `WRL_TYPE` (wallet type) as `file` and `STATUS` as `OPEN`. This confirms that the Oracle wallet has been created.

   NOTE: Re-executing the command "`alter system set encryption key identified by "asdf1234";`" rotates (re-keys) the master encryption key. Every time this command is executed, a new encryption key is generated and stored in Oracle wallet, and the data is encrypted with the new encryption key.

Encrypting a Column in a Table

To encrypt a column in a table, execute the following command:

```
SQL> alter table <table_owner>.<table_name> modify (<column_name> encrypt);
```

Here <table_owner> represents the name of the table owner and <table_name> represents the name of the table containing the column <column_name> to be encrypted.

For example:

```
SQL> alter table appowner.credit_cards modify (ccnum encrypt);
```

To verify whether the encrypted data is accessible, execute the following command:

```
SQL> select * from <table_owner>.<table_name>;
```

For example:

```
SQL> select * from appowner.credit_cards;
```

Opening the Wallet Manually

To open the wallet, execute:

```
alter system set wallet open identified by "asdf1234";
```

Checking the Wallet Status

To check the wallet status, execute:

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

Closing the Wallet

After encrypting the column, close the wallet by executing:

```
SQL> alter system set wallet close identified by "asdf1234";
SQL> exit;
```

Above commands work for both RAC1 and RAC2, after copying the wallet on both instances.

Enabling Auto Login with Oracle Wallet

These steps assume that Oracle TDE is used only in "**FILE**" mode. To enable auto login with Oracle wallet:

1. Make sure that the wallet location in the sqlnet.ora file –on both instances of RAC (for e.g. RAC1, RAC2) is:

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

   Restart the database, if needed
2. Make sure that the wallet is open.

   To check the wallet status, execute:

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

   If the wallet is closed, open the wallet by executing: alter system set wallet open identified by "asdf1234";
3. Navigate to /etc/oracle/wallet/RAC directory.
4. Create a (local) auto-open encryption wallet at /etc/oracle/wallet/RAC. Execute the following command:

```
$ orapki wallet create -wallet . -auto_login
```

Here `.` represents the current directory, which should be `/etc/oracle/wallet/RAC`. Enter this path if the current directory is not `/etc/oracle/wallet/RAC`.

5. Enter the wallet password (`asdf1234`) when prompted.
6. A new file, **cwallet.sso**, is created at the `/etc/oracle/wallet/RAC` directory. This directory now contains two files, **cwallet.sso** and **ewallet.p12**.

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.

7. Close the Oracle wallet. Execute the following command:

```
SQL> alter system set encryption wallet close identified by "asdf1234";
```

8. Open the wallet again. Run the following command:

```
SQL> alter system set encryption wallet open identified by "asdf1234";
```

9. Validate auto login.

   Restart the database, and query a table in an encrypted tablespace. If the query succeeds, then auto-open has been enabled (because the wallet was automatically opened by Oracle without supplying the password).

From now onward, no password is required to access the data encrypted with the master key.

Migrating Oracle Wallet to HSM

An existing Oracle wallet can be migrated to HSM. After the wallet is migrated, auto login can also be enabled with the wallet migrated to HSM.

Migrating an Oracle wallet to HSM and enabling auto login with the migrated wallet involves the following steps:

1. **Migrating an Oracle Wallet to HSM**
2. **Configuring Auto Login with Migrated Wallet**

Migrating an Oracle Wallet to HSM

1. Change the `FILE` to `HSM` in `$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. Restart the database.
3. Execute the following command:

```
SQL> alter system set encryption key identified by "tdeowner:asdf1234" migrate
using "asdf1234";
```

Here "`tdeowner:asdf1234`" represents the NAE user name and its password and "`asdf1234`" represents the password to access the existing Oracle wallet.

The Oracle wallet is now successfully migrated to HSM and a master encryption key is generated on SafeNet KeySecure.

4. Check the wallet status by executing:

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

The command output will display entries for file and HSM wallets each with the `STATUS` as `OPEN`.

5. Verify whether the data encrypted with Oracle wallet is accessible over HSM.

Execute the following command:

```
SQL> select * from <table_owner>.<table_name>;
```

For example:

```
SQL> select * from appowner.credit_cards;
```

6. Once the encryption key is migrated to SafeNet KeySecure, encryption requests similar to the following appear in the activity logs on SafeNet KeySecure every three seconds.

```
[2012-08-07 17:12:01] INFO 192.168.1.22 [-] tdeowner 100003 Crypto
ORACLE.TDE.HSM.MK.06AF95FC1B8BFD4FA0BFFB6D9D68B36AE3 [op#1 ENCRYPT
AES/CBC/PKCS5Padding] - [Success] [-]
```

These encryptions are a result of Oracle's heartbeat functionality related to TDE, specifically, related to external HSMs.

NOTE: After migration of an auto login Oracle wallet to HSM, the auto login functionality is no longer available with the migrated wallet. Auto login needs to be reconfigured with the migrated wallet.

Configuring Auto Login with Migrated Wallet

To configure auto login with wallet migrated to HSM (SafeNet KeySecure):

1. Make sure that the wallet location in the sqlnet.ora file –on both instances of RAC (for e.g. RAC1, RAC2) is:

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

Restart the database, if needed.

2. Navigate to `/etc/oracle/wallet/RAC` directory.

3. Enable auto-open HSM. Add the following entry to the empty wallet by executing:

```
$ mkstore -wrl . -createEntry ORACLE.TDE.HSM.AUTOLOGIN any-non-empty-string
```

Enter the wallet password (`asdf1234`) when prompted.

4. 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;

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.

5. Close the wallet (connection to the HSM). Execute the following command:

```
SQL> alter system set encryption wallet close identified by "tdeowner:asdf1234";
```

NOTE: If the above command results in the error "`ORA-28365: wallet is not open`", it means the wallet is already closed.

6. Open the wallet again. Run the following command:

```
SQL> alter system set encryption wallet open identified by "tdeowner:asdf1234";
```

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.

7. Validate auto login.

   Restart the database, and query a table in an encrypted tablespace. If the query succeeds, then auto-open has been enabled (because the wallet was automatically opened by Oracle).

   From now onward, no password is required to access the encrypted data with the master encryption key.

Managing HSM Wallets

Encryption wallet can be of two types: HSM (SafeNet KeySecure) and Oracle (software) wallets. This section describes instructions to configure an HSM wallet.

Configuring HSM Wallets

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.

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

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

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>
```

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

```
SQL> alter system set encryption key identified by "tdeowner:asdf1234";
```

NOTE:

- In this sample command, `"tdeowner:asdf1234"` represents the NAE user name and its password. The NAE user name and password are case-sensitive. They must appear in double-quotes (") and be separated by a colon (:).
- The NAE user specified in the above command is the owner of the encryption key created and stored on SafeNet KeySecure.
- The Key and Policy Configuration page of SafeNet KeySecure Management Console displays the generated master encryption key.

Encrypting a Column in a Table

After the master encryption key is generated, it can be used to encrypt a column of a table or tablespace. To encrypt a column in a table, execute the following command:

```
SQL> alter table <table_owner>.<table_name> modify (<column_name> encrypt);
```

Here `<table_owner>` represents the name of the table owner and `<table_name>` represents the name of the table containing the column `<column_name>` to be encrypted.

For example:

```
SQL> alter table appowner.credit_cards modify (ccnum encrypt);
```

To verify whether the encrypted data is accessible, execute the following command:

```
SQL> select * from <table_owner>.<table_name>;
```

For example:

```
SQL> select * from appowner.credit_cards;
```

NOTE: On Windows platforms, while performing select operations on the encrypted table, the following error may occur: `ORA-28353: failed to open wallet`. This error occurs because the wallet gets automatically closed after exiting a SQL session. Therefore, the wallet must be opened manually before performing any operations on the encrypted table.

Opening the Wallet Manually

To open the wallet, execute:

```
alter system set wallet open identified by "tdeowner:asdf1234";
```

Checking the Wallet Status

To check the wallet status, execute:

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

Oracle TDE/PKCS#11 can also be used to perform tablespace encryption.

Closing the Wallet

After encrypting a column in a table, close the wallet by executing:

```
SQL> alter system set wallet close identified by "tdeowner:asdf1234";
SQL> exit;
```

IMPORTANT: Oracle TDE/PKCS#11 can also be used to enable auto login wallets and to perform tablespace encryption. For details, see “Enabling Auto Login with HSM” below.

Enabling Auto Login with HSM

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

1. The current entry in sqlnet.ora – file on both instances of RAC (for e.g. RAC1, RAC2):

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

   needs to be changed to:

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

1. 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.

2. 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.

3. 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.

4. Close the connection to the HSM with

```
SQL> alter system set encryption wallet close identified by "tdeowner:asdf1234";
```

And open it one last time with

```
SQL> alter system set encryption wallet open identified by "tdeowner:asdf1234";
```

5. Validate auto login.

   Restart the database, and query a table in an encrypted tablespace. If the query succeeds, then auto-open has been enabled (because the wallet was automatically opened by Oracle without supplying the password).

   From now onwards, no password is required to access the data encrypted with the master key.

TDE Tablespace Encryption

First, open the wallet on RAC1 machine.

```
SQL> alter system set encryption wallet open identified by "tdeowner:asdf1234";
```

Create an encrypted tablespace in the shared disk.

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

Create a table in the tablespace.

```
SQL>create table customer_payment ( first_name varchar2(11),last_name varchar2(11))
TABLESPACE mytablespace;
```

Insert some values in customer_payment table

```
SQL>insert into customer_payment values('Test','Test');

SQL> commit;
```

Close the wallet

```
SQL> alter system set encryption wallet close identified by "tdeowner:asdf1234";
```

Now try to access the customer_payment table.

```
SQL> select * from customer_payment;
```

You will get the following error as the wallet is not open:

```
ORA-28365: wallet is not open
```

Now you can go on RAC 2 machine. Firstly open the wallet on RAC2 machine

```
SQL> alter system set encryption wallet open identified by "tdeowner:asdf1234";
```

Now try to access the customer_payment table

```
SQL> select * from customer_payment;
```

1 row will be listed as entered above in the table.

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.

Managing Multiple Databases on RAC

In case you need to have multiple databases on RAC, then simply create a RAC database using dbca utility and follow the same set of instructions as mentioned above. You need to make sure of the following while working with the new database:

1. *ORACLE_SID's* on both RAC1 & RAC2 are set to new db instance SID respectively, for e.g. if we have two RAC databases *orcl* and *orclnew* then while working with *orclnew* the *ORACLE _SID* must be set to *orclnew1* on RAC1 and *orclnew2* on RAC2. Similarly for *orcl, ORACLE_SID* must be set to *orcl1* on RAC1 and *orcl2* on RAC2.
2. Create separate wallet locations for both the RAC databases, for e.g. for *orcl* database the wallet can be created at **/home/oracle/orcl/WALLET** and for *orclnew* database the wallet should be created at **/home/oracle/orclnew/WALLET**.
3. All the databases, e.g. *orcl* and *orclnew*, need to share the same properties file for configuration purpose.
4. Use of different NAE users is suggested for different databases. All the NAE users should be covered under one group that has privileges to use the TSE key. This is needed as oracle sends a keyinfo request for TSE key.
5. For any new database make sure to change the db name while running the *export* and *srvctl* commands.

9

Managing Multiple Databases on a Single Oracle Instance

Prerequisites

Oracle Database

Multiple Oracle Database 11g (or higher) must be installed on the target machine with unique SID's to carry on with the integration process. Refer to the Oracle documentation for detailed installation instructions for the database and the Transparent Data Encryption feature.

NOTE:

- This document assumes orcl1 & orcl2 as the two database unique SID's.
- All the databases, e.g. orcl1 and orcl2, need to share the same properties file for configuration purpose.

Configuring Multiple Databases

In order to work with multiple databases on a single oracle instance, follow the below mentioned steps:

1. Refer to Chapter 2 & 3 for configuring SafeNet PKCS#11 on Linux/UNIX and Windows.
2. Configuring HSM Wallets
3. Configuring Oracle Wallets

Configuring HSM Wallets

After configuring SafeNet PKCS#11 with Oracle TDE, the master key for TDE can be generated and stored on HSM. Generation of the master key involves the following steps:

a. Adding the Wallet Location to sqlnet.ora
b. Restarting the Oracle Database
c. Creating the Encryption Wallet
d. Encrypting a Column in a Table
e. Closing the Wallet

NOTE: For each of the two databases i.e. orcl1 & orcl2, run the below export command on two different consoles:

- Export ORACLE_SID=orcl1
- Export ORACLE_SID=orcl2

Adding the Wallet Location to sqlnet.ora

On Linux/UNIX

Add the following code to the $ORACLE_HOME/network/admin/sqlnet.ora file.

```
ENCRYPTION_WALLET_LOCATION =
(SOURCE =
(METHOD = HSM)
(METHOD_DATA = (DIRECTORY = <desired_path>)
)
```

On Windows

Add the following to $ORACLE_HOME\NETWORK\ADMIN\sqlnet.ora.

```
ENCRYPTION_WALLET_LOCATION =
(SOURCE =
(METHOD = HSM)
(METHOD_DATA = (DIRECTORY = %SYSTEM_DRIVE%\<desired_path>))
```

NOTE:

- It is recommended to use an HSM rather than Oracle wallet. However, if an Oracle wallet is to be used, set (METHOD = FILE) instead of HSM. For details, see Chapter 6, “Managing Oracle Wallets”.
- In the above code, <desired_path> specifies the location where the wallet will be stored. Make sure that <desired_path> specified by the DIRECTORY parameter exists and the Oracle user has appropriate permissions on it. From this point onward, this document uses /home/oracle/$ORACLE_SID/WALLET as <desired_path> (the wallet location).

Restarting the Oracle Database

After making any changes to the sqlnet.ora file, it is necessary to restart the database for the changes to be effective. To restart the database, execute the following commands:

```
$ sqlplus /nolog
SQL> connect / as sysdba
Connected
```

```
SQL> shutdown immediate
Database closed.
Database dismounted.
ORACLE instance shut down.
SQL> startup
ORACLE instance started.
Total System Global Area 167772160 bytes
Fixed Size …… 1247900 bytes
Variable Size …… 75498852 bytes
Database Buffers …… 88080384 bytes
Redo Buffers …… 2945024 bytes
Database mounted.
Database opened.
SQL> exit
$ lsnrctl stop
$ lsnrctl start
```

Creating the Encryption Wallet

After the database is restarted, the encryption wallet can be created to store the master encryption keys.

To create an encryption wallet (if it does not already exist at the location specified in the sqlnet.ora file.) and add a master encryption key to it, execute the following commands:

NOTE: To be performed on both orcl1 & orcl2.

```
$ sqlplus system/<system_password>
SQL> alter system set encryption key identified by "tdeowner:asdf1234";
```

NOTE:

- In this sample command, "tdeowner:asdf1234" represents the NAE user name and its password. The NAE user name and password are case-sensitive. They must appear in double-quotes (") and be separated by a colon (:).
- The NAE user, specified in the above command, is the owner of the encryption key created and stored on SafeNet KeySecure.
- The Key and Policy Configuration page of SafeNet KeySecure Management Console displays the generated master encryption key.
- Both orcl1 and orcl2 should use different NAE users.

TIP: Re-executing the command “alter system set encryption key identified by **"tdeowner:asdf1234";**” rotates (re-keys) the master encryption key. Every time this command is executed, a new encryption key is generated and stored on SafeNet KeySecure, and the data is encrypted with the new encryption key.

NOTE: While creating wallet for two different databases, use two separate NAE users and those should be covered under one group that has privileges to use the TSE key for tablespace encryption.

Encrypting a Column in a Table

NOTE: To be performed on both orcl1 & orcl2.

After the master encryption key is generated, it can be used to encrypt a column of a table or tablespace. To encrypt a column in a table, execute the following command:

```
SQL> alter table <table_owner>.<table_name> modify (<column_name> encrypt);
```

Here <table_owner> represents the name of the table owner and <table_name> represents the name of the table containing the column <column_name> to be encrypted.

For example:

```
SQL> alter table appowner.credit_cards modify (ccnum encrypt);
```

To verify whether the encrypted data is accessible, execute the following command:

```
SQL> select * from <table_owner>.<table_name>;
```

For example:

```
SQL> select * from appowner.credit_cards;
```

NOTE: On Windows platforms, while performing select operations on the encrypted table, the following error may occur: `ORA-28353: failed to open wallet`. This error occurs because the wallet gets automatically closed after exiting a SQL session. Therefore, the wallet must be opened manually before performing any operations on the encrypted table.

Opening the Wallet Manually

To open the wallet, execute:

```
alter system set wallet open identified by "tdeowner:asdf1234";
```

Checking the Wallet Status

To check the wallet status, execute:

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

Closing the Wallet

After encrypting a column in a table, close the wallet by executing:

```
SQL> alter system set wallet close identified by "tdeowner:asdf1234";
SQL> exit;
```

IMPORTANT: Oracle TDE/PKCS#11 can also be used to enable auto login wallets and to perform tablespace encryption. For details, see “Enabling Auto Login Wallet” below.

Enabling Auto Login Wallet

NOTE: To be performed on both databases i.e. orcl1 & orcl2.

When an encrypted wallet is password-protected, then a valid password is needed to open the wallet. After the wallet is opened, the authorized users and applications can perform operations on the encrypted data.

To eliminate the need for supplying password, auto-open (auto login) wallet can be created. With auto-open wallet, the encrypted wallet is automatically opened as soon as the database is started. Authorized users and applications can access the encrypted data without supplying password.

NOTE: An auto-open wallet should be created from an existing encryption wallet. This ensures that the master key can be transferred to the new auto-open wallet.

WARNING: Do not delete the encryption wallet after creating an auto-open wallet. The encryption wallet is needed for rotation (or re-key) of the master encryption key.

Enabling Auto Login with HSM

These steps assume that Oracle TDE is used only in “HSM” mode, i.e., it is never migrated from an Oracle wallet.

To enable auto login with HSM:

1. Make sure that the wallet is open.

   To check the wallet status, execute:

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

   If the wallet is closed, open the wallet by executing:

   ```
   alter system set wallet open identified by "tdeowner:asdf1234";
   ```

2. Modify the existing sqlnet.ora file. Add the following code, if needed:

   ```
   (METHOD_DATA = (DIRECTORY =/home/oracle/$ORACLE_SID/WALLET))
   ```

Make sure that the path specified by the DIRECTORY parameter exists and the Oracle user has appropriate permissions on it.

The content in the updated sqlnet.ora file should look similar to the following:

```
ENCRYPTION_WALLET_LOCATION =

(SOURCE = (METHOD = HSM)

(METHOD_DATA = (DIRECTORY =/home/oracle/$ORACLE_SID/WALLET)))
```

3. Navigate to /home/oracle/$ORACLE_SID/WALLET directory.
4. Create a (local) auto-open encryption wallet at /home/oracle/$ORACLE_SID/WALLET. Execute the following command:

   ```
   $ orapki wallet create -wallet . -auto_login
   ```

   Here **.** represents the current directory, which should be /home/oracle/$ORACLE_SID/WALLET. Enter this path if the current directory is not /home/oracle/$ORACLE_SID/WALLET.

   Enter and confirm a password for the wallet when prompted.

   This directory now contains two files, cwallet.sso and ewallet.p12.
5. Enable auto-open HSM. Add the following entry to the empty wallet by executing:

   ```
   $ mkstore -wrl . -createEntry ORACLE.TDE.HSM.AUTOLOGIN any-non-empty-string
   ```

   Enter the wallet password (created in the previous step) when prompted.
6. Close the wallet (connection to the HSM). Execute the following command:

   ```
   SQL> alter system set encryption wallet close identified by "tdeowner:asdf1234";
   ```

   NOTE: Do not delete the encryption wallet after creating an auto-open wallet. The encryption wallet is needed for rotation (or re-key) of the master encryption key.
7. Open the wallet again. Run the following command:

   ```
   SQL> alter system set encryption wallet open identified by "tdeowner:asdf1234";
   ```
8. Validate auto login.

   Restart the database, and query a table in an encrypted tablespace. If the query succeeds, then auto-open has been enabled (because the wallet was automatically opened by Oracle without supplying the password).

   From now onward, no password is required to access the data encrypted with the master key.

Configuring Oracle Wallets

NOTE: Ensure that the **ORACLE_SID** is set to orcl1 on one console and orcl2 on the other.

After configuring SafeNet PKCS#11 with Oracle TDE, the encryption key for TDE can be generated and stored in the Oracle wallet (i.e., in database). Generation of the encryption key involves the following steps:

a. Adding the Wallet Location to sqlnet.ora

b. Restarting the Oracle Database

c. Creating the Encryption Wallet
d. Encrypting a Column in a Table
e. Closing the Wallet

Adding the Wallet Location to sqlnet.ora

NOTE: To be performed on both databases i.e. orcl1 & orcl2.

On Linux/UNIX

Add the following code to the $ORACLE_HOME/network/admin/sqlnet.ora file.

```
ENCRYPTION_WALLET_LOCATION =
(SOURCE =
(METHOD = FILE)
(METHOD_DATA = (DIRECTORY = <desired_path>)
)
```

On Windows

Add the following to $ORACLE_HOME\NETWORK\ADMIN\sqlnet.ora.

```
ENCRYPTION_WALLET_LOCATION =
(SOURCE =
(METHOD = FILE)
(METHOD_DATA = (DIRECTORY = %SYSTEM_DRIVE%\<desired_path>))
)
```

NOTE: In the above code, <desired_path> specifies the location where the wallet will be stored. Make sure that <desired_path> specified by the DIRECTORY parameter exists and the Oracle user has appropriate permissions on it. From this point onward, this document uses **/home/oracle/$ORACLE_SID/WALLET** as **<desired_path>** (the wallet location).

Restarting the Oracle Database

After making any changes to the sqlnet.ora file, it is necessary to restart the database for the changes to be effective. To restart the database, execute the following commands:

```
$ sqlplus /nolog
SQL> connect / as sysdba
Connected
SQL> shutdown immediate
Database closed.
```

```
Database dismounted.
ORACLE instance shut down.
SQL> startup
ORACLE instance started.
Total System Global Area 167772160 bytes
Fixed Size ..... 1247900 bytes
Variable Size ..... 75498852 bytes
Database Buffers ..... 88080384 bytes
Redo Buffers ..... 2945024 bytes
Database mounted.
Database opened.
SQL> exit
$ lsnrctl stop
$ lsnrctl start
```

Creating the Encryption Wallet

NOTE: To be performed on both databases i.e. orcl1 & orcl2.

After restarting the database, the encryption wallet can be created to store the encryption keys.

To create an encryption wallet (at the location specified in the sqlnet.ora file) and add a master encryption key to it,

1. Execute the following command:

   ```
   $ sqlplus system/<system_password>
   ```

   At this point, no Oracle wallet exists. This can be confirmed by executing:

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

   The command output will display WRL_TYPE (wallet type) as file and STATUS as CLOSED.

2. Execute the following command:

   ```
   SQL> alter system set encryption key identified by "asdf1234";
   ```

NOTE:

- In this sample command, "asdf1234" represents the password to access the Oracle wallet. The password is case-sensitive and must appear in double-quotes (").
- As the encryption key is stored in Oracle wallet, the command requires only the password, not the NAE user name. However, in case of HSM wallet, where the encryption key is created and stored on HSM (SafeNet KeySecure), the command requires both the NAE user name (the key

owner) and its password.

- The above command creates the ewallet.p12 file at /home/oracle/$ORACLE_SID/WALLET.

3. Confirm the wallet status. Execute the following command:

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

4. The command output will display WRL_TYPE (wallet type) as file and STATUS as OPEN. This confirms that the Oracle wallet has been created.

TIP: Re-executing the command “alter system set encryption key identified by "asdf1234";” rotates (re-keys) the master encryption key. Every time this command is executed, a new encryption key is generated and stored in Oracle wallet, and the data is encrypted with the new encryption key.

Encrypting a Column in a Table

To encrypt a column in a table, execute the following command:

```
SQL> alter table <table_owner>.<table_name> modify (<column_name> encrypt);
```

Here <table_owner> represents the name of the table owner and <table_name> represents the name of the table containing the column <column_name> to be encrypted.

For example:

```
SQL> alter table appowner.credit_cards modify (ccnum encrypt);
```

To verify whether the encrypted data is accessible, execute the following command:

```
SQL> select * from <table_owner>.<table_name>;
```

For example:

```
SQL> select * from appowner.credit_cards;
```

NOTE: On Windows platforms, while performing select operations on the encrypted table, the following error may occur: **`ORA-28353: failed to open wallet`**. This error occurs because the wallet gets automatically closed after exiting a SQL session. Therefore, the wallet must be opened manually before performing any operations on the encrypted table.

Opening the Wallet Manually

To open the wallet, execute:

```
alter system set wallet open identified by "asdf1234";
```

Checking the Wallet Status

To check the wallet status, execute:

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

Closing the Wallet

After encrypting a column in a table, close the wallet by executing:

```
SQL> alter system set wallet close identified by "asdf1234";
SQL> exit;
```

Enabling Auto Login Wallet

When an encrypted wallet is password-protected, then a valid password is needed to open the wallet. After the wallet is opened, the authorized users and applications can perform operations on the encrypted data.

To eliminate the need for supplying password, auto-open (Auto Login) wallet can be created. With auto-open wallet, the encrypted wallet is automatically opened as soon as the database is started. Authorized users and applications can access the encrypted data without supplying password.

NOTE: An auto-open wallet should be created from an existing encryption wallet. This ensures that the master key can be transferred to the new auto-open wallet.

WARNING: Do not delete the encryption wallet after creating an auto-open wallet. The encryption wallet is needed for rotation (or re-key) of the master encryption key.

Enabling Auto Login with Oracle Wallet

These steps assume that Oracle TDE is used only in “FILE” mode.

To enable auto login with Oracle wallet:

1. Make sure that the wallet is open.

   To check the wallet status, execute:

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

   If the wallet is closed, open the wallet by executing:

   ```
   alter system set wallet open identified by "asdf1234";
   ```

2. Modify the existing sqlnet.ora file. Add the following code, if needed:

   ```
   (METHOD_DATA = (DIRECTORY =/home/oracle/$ORACLE_SID/WALLET)))
   ```

   Make sure that the path specified by the DIRECTORY parameter exists and the Oracle user has appropriate permissions on it.

   The content in the updated sqlnet.ora file should look similar to the following:

   ```
   ENCRYPTION_WALLET_LOCATION =
   (SOURCE = (METHOD = FILE)
   (METHOD_DATA = (DIRECTORY =/home/oracle/$ORACLE_SID/WALLET)))
   ```

3. Restart the database to make any changes in the sqlnet.ora file to be effective. This step is required only if the sqlnet.ora file is modified.
4. Navigate to /home/oracle/$ORACLE_SID/WALLET directory.
5. Create a (local) auto-open encryption wallet at /home/oracle/$ORACLE_SID /WALLET. Execute the following command:

   ```
   $ orapki wallet create -wallet . -auto_login
   ```

   Here . represents the current directory, which should be /home/oracle/$ORACLE_SID /WALLET.

   Enter this path if the current directory is not /home/oracle/$ORACLE_SID WALLET.

   Enter and confirm a password (asdf1234) for the wallet when prompted.

   A new file, cwallet.sso, is created at the /home/oracle/$ORACLE_SID /WALLET directory. This directory now contains two files, cwallet.sso and ewallet.p12.
6. Close the wallet (connection to the HSM). Execute the following command:

   ```
   SQL> alter system set encryption wallet close identified by "asdf1234";
   ```
7. Open the wallet again. Run the following command:

   ```
   SQL> alter system set encryption wallet open identified by "asdf1234";
   ```
8. Validate auto login.

   Restart the database, and query a table in an encrypted tablespace. If the query succeeds, then auto-open has been enabled (because the wallet was automatically opened by Oracle without supplying the password).

   From now onward, no password is required to access the data encrypted with the master key.

Migrating Oracle Wallet to HSM

An existing Oracle wallet can be migrated to HSM. After the wallet is migrated, auto login can also be enabled with the wallet migrated to HSM.

Migrating an Oracle wallet to HSM and enabling auto login with the migrated wallet involves the following steps:

1. **Migrating an Oracle Wallet to HSM**
2. **Configuring Auto Login with Migrated Wallet**

Migrating an Oracle Wallet to HSM

The steps described below assume that an auto login Oracle wallet is already created as described in “Configuring Oracle Wallets” on page 73 and “Enabling Auto Login Wallet” on page 77.

To migrate an Oracle wallet to HSM:

1. Modify the **sqlnet.ora** file, as follows:

   ```
   Change (SOURCE = (METHOD = FILE) to (SOURCE = (METHOD = HSM).
   ```

The content in the updated sqlnet.ora file should look similar to the following:

```
ENCRYPTION_WALLET_LOCATION =
(SOURCE = (METHOD = HSM)
(METHOD_DATA = (DIRECTORY =/home/oracle/$ORACLE_SID/WALLET)))
```

2. Restart the database.
3. Execute the following command:

```
SQL> alter system set encryption key identified by "tdeowner:asdf1234" migrate
using "asdf1234";
```

Here "**tdeowner:asdf1234**" represents the NAE user name and its password and "**asdf1234**" represents the password to access the existing Oracle wallet.

The Oracle wallet is now successfully migrated to HSM and a master encryption key is generated on SafeNet KeySecure.

4. Check the wallet status by executing:

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

The command output will display entries for file and HSM wallets each with the STATUS as OPEN.

5. Verify whether the data encrypted with Oracle wallet is accessible over HSM. Execute the following command:

```
SQL> select * from <table_owner>.<table_name>;
```

For example:

```
SQL> select * from appowner.credit_cards;
```

Once the encryption key is migrated to SafeNet KeySecure, encryption requests similar to the following appear in the activity logs on SafeNet KeySecure every three seconds.

```
[2012-08-07 17:12:01] INFO 192.168.1.22 [-] tdeowner 100003 Crypto
ORACLE.TDE.HSM.MK.06AF95FC1B8BFD4FA0BFFB6D9D68B36AE3 [op#1 ENCRYPT
AES/CBC/PKCS5Padding] - [Success] [-]
```

These encryptions are a result of Oracle's heartbeat functionality related to TDE, specifically, related to external HSMs.

NOTE: After migration of an auto login Oracle wallet to HSM, the auto login functionality is no longer available with the migrated wallet. Auto login needs to be reconfigured with the migrated wallet.

For instructions, see the next topic.

Configuring Auto Login with Migrated Wallet

To configure auto login with wallet migrated to HSM (SafeNet KeySecure):

1. Make sure that the wallet location in the sqlnet.ora file is:

```
ENCRYPTION_WALLET_LOCATION =

(SOURCE = (METHOD = HSM)

(METHOD_DATA = (DIRECTORY =/home/oracle/$ORACLE_SID/WALLET)))
```

Restart the database, if needed.

2. Navigate to **/home/oracle/$ORACLE_SID/WALLET** directory.
3. Enable auto-open HSM. Add the following entry to the empty wallet by executing:

```
$ mkstore -wrl . -createEntry ORACLE.TDE.HSM.AUTOLOGIN any-non-empty-string
```

Enter the wallet password (asdf1234) when prompted.

4. Close the wallet (connection to the HSM). Execute the following command:

```
SQL> alter system set encryption wallet close identified by "tdeowner:asdf1234";
```

> **NOTE:** If the above command results in the error "**`ORA-28365: wallet is not open`**", it means the wallet is already closed.

5. Open the wallet again. Run the following command:

```
SQL> alter system set encryption wallet open identified by "tdeowner:asdf1234";
```

6. Validate auto login.

   Restart the database, and query a table in an encrypted tablespace. If the query succeeds, then auto-open has been enabled (because the wallet was automatically opened by Oracle).

From now onward, no password is required to access the encrypted data with the master encryption key.

10

Integrating TDE with SafeNet KeySecure on Oracle 12c (12.1.0.2 – 64 bit)

This chapter contains the following information:

Setting up SafeNet KeySecure for Transparent Data Encryption (TDE) with Oracle 12c

Working with Pluggable Databases (PDB)

Setting up SafeNet KeySecure for Transparent Data Encryption (TDE) with Oracle 12c

NOTE: Please refer Chapter 2 for Configuring SafeNet PKCS#11 on Linux/UNIX.

Managing HSM Wallets

Encryption wallet can be of two types: HSM (SafeNet KeySecure) and Oracle (software) wallets. It is recommended to use an HSM rather than Oracle wallet.

NOTE: This section describes steps to perform fresh configuration of an HSM wallet. The steps described assume that no HSM or Oracle wallet already exists. For information on configuring Oracle wallets, see “Managing Oracle Wallets”.

Configuring HSM Wallets

1. **Set the Hardware Keystore Type in the sqlnet.ora File**
2. **Open the Hardware Keystore**
3. **Set the Hardware Keystore TDE Master Encryption Key**
4. **Encrypt Your Data**
5. **Close the Hardware Keystore**

Set the Hardware Keystore Type in the sqlnet.ora File

Before you can configure a hardware keystore, you must enable the database to recognize that it is a hardware security module by editing the `sqlnet.ora` file. By default, this file is located in the `ORACLE_HOME/network/admin` directory or in the location set by the `TNS_ADMIN` environment variable.

Use the following setting in the `sqlnet.ora` file to define the hardware keystore type, which is HSM.

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

Open the Hardware Keystore

1. Log in to the database instance as a user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.

   For example, to log in to the root:

   ```
   sqlplus / as sysdba
   connected to: Oracle Database 12c
   SQL> GRANT ADMINISTER KEY MANAGEMENT to system;
   Grant succeeded.
   SQL> commit;
   Commit complete.
   SQL> Connect system/<system_password>
   Connected.
   ```

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

   ```
   ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN IDENTIFIED BY "user_id:password";
   ```

   For example:

   ```
   ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN IDENTIFIED BY "tdeowner:asdf1234";
   keystore altered.
   ```

   NOTE:

   - In this sample command, `"tdeowner:asdf1234"` represents the NAE user name and its password. The NAE user name and password are case-sensitive. They must appear in double-quotes (") and be separated by a colon (:).
   - The Key and Policy Configuration page of SafeNet KeySecure Management Console displays the generated master encryption key.

3. Repeat this procedure each time you restart the database instance.

Set the Hardware Keystore TDE Master Encryption Key

1. Run the following SQL statement:

   ```
   ADMINISTER KEY MANAGEMENT SET KEY IDENTIFIED BY "user_id:password";
   ```

For example:

```
ADMINISTER KEY MANAGEMENT SET KEY IDENTIFIED BY "tdeowner:asdf1234";

keystore altered.
```

NOTE:

- In this sample command, `"tdeowner:asdf1234"` represents the NAE user name and its password. The NAE user name and password are case-sensitive. They must appear in double-quotes (") and be separated by a colon (:).
- The NAE user specified in the above command is the owner of the encryption key created and stored on SafeNet KeySecure.
- The Key and Policy Configuration page of SafeNet KeySecure Management Console displays the generated master encryption key.

TIP: Re-executing the command "`ADMINISTER KEY MANAGEMENT SET KEY IDENTIFIED BY "tdeowner:asdf1234";`" rotates (re-keys) the master encryption key. Every time this command is executed, a new encryption key is generated and stored on SafeNet KeySecure, and the data is encrypted with the new encryption key.

Encrypt Your Data

At this stage, the keystore configuration is complete and you can begin to encrypt data.

Encrypting a Column in a Table

After the master encryption key is generated, it can be used to encrypt a column of a table or tablespace. To encrypt a column in a table, execute the following command:

```
SQL> alter table <table_owner>.<table_name> modify (<column_name> encrypt);
```

Here <table_owner> represents the name of the table owner and <table_name> represents the name of the table containing the column <column_name> to be encrypted.

For example:

```
SQL> alter table appowner.credit_cards modify (ccnum encrypt);
```

To verify whether the encrypted data is accessible, execute the following command:

```
SQL> select * from <table_owner>.<table_name>;
```

For example:

```
SQL> select * from appowner.credit_cards;
```

Creating an Encrypted Tablespace

Run the CREATE TABLESPACE statement, using its encryption clauses.

For example:

```
CREATE TABLESPACE encrypt_ts DATAFILE '$ORACLE_HOME/dbs/encrypt_df.dbf' SIZE 1M
ENCRYPTION USING 'AES256' DEFAULT STORAGE (ENCRYPT);
```

In this specification:

- `ENCRYPTION USING 'AES256'` specifies the encryption algorithm and the key length for the encryption. Enclose this setting in single quotation marks (' '). The key lengths are included in the names of the algorithms. If you do not specify an encryption algorithm, then the default encryption algorithm, AES128, is used. Choose from the following algorithms:
- 3DES168
- AES128
- AES192
- AES256
- `ENCRYPT` in the `DEFAULT STORAGE` clause encrypts the tablespace.

Close the Hardware Keystore

After encrypting a column in a table or a tablespace, close the wallet by executing:

```
SQL> ADMINISTER KEY MANAGEMENT SET KEYSTORE CLOSE IDENTIFIED BY
"tdeowner:asdf1234";

SQL> exit;
```

Enabling Auto Login with HSM

1. Ensure that you configured the TDE hardware keystore.
2. Close the hardware security module if it is open. (You can check the status of whether a keystore is open or closed by querying the `STATUS` column of the `V$ENCRYPTION_WALLET` view.)

   For example:

   ```
   ADMINISTER KEY MANAGEMENT SET KEYSTORE CLOSE IDENTIFIED BY "tdeowner:password";
   ```
3. Reconfigure the `sqlnet.ora` file and add the keystore location of the software keystore created in Step 3 to the `DIRECTORY` setting of the `ENCRYPTION_WALLET_LOCATION` setting.

   For example:

   ```
   ENCRYPTION_WALLET_LOCATION=
    (SOURCE=(METHOD=FILE)(METHOD_DATA=
      (DIRECTORY=/etc/ORACLE/WALLETS/orcl)))
   ```
4. Restart the `sqlplus` session.
5. If you have not migrated from a software keystore, then create the software keystore with the hardware keystore password in the appropriate location (for example, `/etc/ORACLE/WALLETS/orcl`).

   For example:

   ```
   ADMINISTER KEY MANAGEMENT CREATE KEYSTORE '/etc/ORACLE/WALLETS/orcl' IDENTIFIED BY
   "tdeowner:password";
   ```
6. Open the software keystore.

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

   For example:

```
ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN IDENTIFIED BY "tdeowner:asdf1234";
```

7. Add the secret in the software keystore.

   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.

   ```
   ADMINISTER KEY MANAGEMENT ADD SECRET "user_id:password" FOR CLIENT "HSM_PASSWORD"
   IDENTIFIED BY software_keystore_password WITH BACKUP;
   ```

   For example:

   ```
   ADMINISTER KEY MANAGEMENT ADD SECRET 'tdeowner:asdf1234' FOR CLIENT 'HSM_PASSWORD'
   IDENTIFIED BY "tdeowner:asdf1234" WITH BACKUP;
   ```

8. Close the software keystore.

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

   For example:

   ```
   ADMINISTER KEY MANAGEMENT SET KEYSTORE CLOSE IDENTIFIED BY "tdeowner:asdf1234";
   ```

9. Create (or re-create) the auto-login keystore.

   ```
   ADMINISTER KEY MANAGEMENT CREATE AUTO_LOGIN KEYSTORE FROM KEYSTORE
   '/etc/ORACLE/WALLETS/orcl' IDENTIFIED BY "tdeowner:asdf1234";
   ```

10. A new file, **cwallet.sso**, is created at the `/etc/ORACLE/WALLETS/orcl` directory. This directory now contains two files, **cwallet.sso** and **ewallet.p12**.

NOTE: Rename the **ewallet.p12** to **ewallet.p24.**

11. Update the `sqlnet.ora` file to use the hardware security module location.

    For example:

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

    At this stage, the next time that a TDE operation executes, the hardware security module auto-login keystore opens automatically.

Update the secret in the HSM Wallet

The user may require change of password for HSM wallet on regular basis to enhance wallet security. To update the password follow the steps below:

1. Change the `sqlnet.ora` file to FILE.

   For example:

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

2. Restart sqlplus session.
3. Open the wallet using old credentials.

```
ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN IDENTIFIED BY "tdeowner:asdf1234";
```

4. Now use the update command to update the secret.

```
ADMINISTER KEY MANAGEMENT UPDATE SECRET 'tdewoner:asdf12345' FOR CLIENT
'HSM_PASSWORD' IDENTIFIED BY "tdewoner:asdf1234" WITH BACKUP;
```

5. Change the sqlnet.ora file to HSM.

   For example:

```
ENCRYPTION_WALLET_LOCATION=

  (SOURCE=(METHOD=HSM)(METHOD_DATA=

    (DIRECTORY=/etc/ORACLE/WALLETS/orcl)))
```

6. Restart sqlplus session.

Managing Oracle Wallets

Encryption wallet can be of two types: HSM (SafeNet KeySecure) and Oracle (software) wallets. This chapter describes instructions to configure an Oracle wallet. The chapter contains detailed steps to enable auto login with an Oracle wallet and steps to migrate keys from an existing Oracle wallet to HSM.

NOTE: The steps described in this chapter assume that no HSM or Oracle wallet already exists. For information on configuring HSM wallets, see “Managing HSM Wallets”.

Configuring Oracle Wallets

1. **Set the Software Keystore Location in the sqlnet.ora File**
2. **Create the Software Keystore**
3. **Open the Software Keystore**
4. **Set the Software TDE Master Encryption Key**
5. **Encrypt Your Data**
6. **Close the Software Keystore**

Set the Software Keystore Location in the sqlnet.ora File

Oracle Database checks the `sqlnet.ora` file for the directory location of the keystore, whether it is a software keystore or a hardware module security (HSM) keystore. Your first step is to edit the `sqlnet.ora` file to define a directory location for the keystore that you plan to create. Ensure that this directory exists beforehand. Preferably, this directory should be empty.

In a multitenant environment, the keystore location is set for the entire multitenant container database (CDB), not for individual pluggable databases (PDBs).

By default, the sqlnet.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 sqlnet.ora file.

To create a software keystore on a regular file system, use the following format when you edit the sqlnet.ora file:

```
ENCRYPTION_WALLET_LOCATION=
 (SOURCE=
  (METHOD=FILE)
   (METHOD_DATA=
    (DIRECTORY=path_to_keystore)))
```

For Example:

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

Create the Software Keystore

After you have specified a directory location for the software keystore, you can create the keystore.

1. Log in to the database instance as a user who has been granted the ADMINISTER KEY MANAGEMENT or SYSKM privilege.

   For example, to log in to the root:

   ```
   sqlplus / as sysdba

   connected to: Oracle Database 12c

   SQL> GRANT ADMINISTER KEY MANAGEMENT to system;

   Grant succeeded.

   SQL> commit;

   Commit complete.

   SQL> Connect system/<system_password>

   Connected.
   ```

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

   The syntax is as follows:

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

   In this specification:

   - keystore_location is the path to the keystore directory location of the password-based keystore for which you want to create the auto-login keystore (for example, /etc/ORACLE/WALLETS/orcl). Enclose the keystore_location setting in single quotation marks (' '). To find this location, you can query the

WRL_PARAMETER column of the V$ENCRYPTION_WALLET view. (If the keystore was not created in the default location, then the STATUS column of the V$ENCRYPTION_WALLET view is NOT_AVAILABLE.)

- software_keystore_password is the password of the keystore that you, the security administrator, creates.

For example, to create the keystore in the /etc/ORACLE/WALLETS/orcl directory:

```
ADMINISTER KEY MANAGEMENT CREATE KEYSTORE '/etc/ORACLE/WALLETS/orcl' IDENTIFIED BY
"asdf1234";

keystore altered.
```

After you run this statement, the **ewallet.p12** file, which is the keystore, appears in the keystore location.

Open the Software Keystore

1. Log in to the database instance as a user who has been granted the ADMINISTER KEY MANAGEMENT or SYSKM privilege.

   For example, to log in to the root:

   ```
   sqlplus / as sysdba

   connected to: Oracle Database 12c

   SQL> GRANT ADMINISTER KEY MANAGEMENT to system;

   Grant succeeded.

   SQL> commit;

   Commit complete.

   SQL> Connect system/<system_password>

   Connected.
   ```

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

   ```
   ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN IDENTIFIED BY
   "software_keystore_password";
   ```

   For example:

   ```
   ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN IDENTIFIED BY "asdf1234";

   keystore altered.
   ```

Set the Software TDE Master Encryption Key

1. Run the following SQL statement:

   ```
   ADMINISTER KEY MANAGEMENT SET KEY IDENTIFIED BY "password" WITH BACKUP USING
   'backup_identifier';
   ```

   For example:

   ```
   ADMINISTER KEY MANAGEMENT SET KEY IDENTIFIED BY "asdf1234" WITH BACKUP USING
   'enc_key_backup';

   keystore altered.
   ```

Encrypt Your Data

At this stage, the keystore configuration is complete and you can begin to encrypt data.

Encrypting a Column in a Table

After the master encryption key is generated, it can be used to encrypt a column of a table or tablespace. To encrypt a column in a table, execute the following command:

```
SQL> alter table <table_owner>.<table_name> modify (<column_name> encrypt);
```

Here `<table_owner>` represents the name of the table owner and `<table_name>` represents the name of the table containing the column `<column_name>` to be encrypted.

For example:

```
SQL> alter table appowner.credit_cards modify (ccnum encrypt);
```

To verify whether the encrypted data is accessible, execute the following command:

```
SQL> select * from <table_owner>.<table_name>;
```

For example:

```
SQL> select * from appowner.credit_cards;
```

Creating an Encrypted Tablespace

Run the CREATE TABLESPACE statement, using its encryption clauses.

For example:

```
CREATE TABLESPACE encrypt_ts DATAFILE '$ORACLE_HOME/dbs/encrypt_df.dbf' SIZE 1M
ENCRYPTION USING 'AES256' DEFAULT STORAGE (ENCRYPT);
```

In this specification:

- `ENCRYPTION USING 'AES256'` specifies the encryption algorithm and the key length for the encryption. Enclose this setting in single quotation marks (' '). The key lengths are included in the names of the algorithms. If you do not specify an encryption algorithm, then the default encryption algorithm, AES128, is used. Choose from the following algorithms:
- 3DES168
- AES128
- AES192
- AES256
- `ENCRYPT` in the `DEFAULT STORAGE` clause encrypts the tablespace.

Close the Software Keystore

After encrypting a column in a table or a tablespace, close the wallet by executing:

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

For example:

```
ADMINISTER KEY MANAGEMENT SET KEYSTORE CLOSE IDENTIFIED BY "asdf1234";
```

Enabling Auto Login with Oracle Wallet

When an encrypted wallet is password-protected, then a valid password is needed to open the wallet. After the wallet is opened, the authorized users and applications can perform operations on the encrypted data.

To eliminate the need for supplying password, auto-open (Auto Login) wallet can be created. With auto-open wallet, the encrypted wallet is automatically opened as soon as the database is started. Authorized users and applications can access the encrypted data without supplying password.

NOTE: An auto-open wallet should be created from an existing encryption wallet. This ensures that the master key can be transferred to the new auto-open wallet.

WARNING: Do not delete the encryption wallet after creating an auto-open wallet. The encryption wallet is needed for rotation (or re-key) of the master encryption key.

The below steps assume that Oracle TDE is used only in "`FILE`" mode. To enable auto login with Oracle wallet:

1. Open the password-based software keystore

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

   For example:

   ```
   ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN IDENTIFIED BY "asdf1234";
   ```

2. Run the `ADMINISTER KEY MANAGEMENT SQL` statement to create the auto-login keystore.

   The syntax is as follows:

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

   In this specification:

   - `LOCAL` enables you to create a local auto-login software keystore.
   - keystore_location is the path to the directory location of the password-based keystore for which you want to create the auto-login keystore (for example, `/etc/ORACLE/WALLETS/orcl`). Enclose this setting in single quotation marks (' '). To find this location, query the `WRL_PARAMETER` column of the `V$ENCRYPTION_WALLET` view.
   - software_keystore_password is the password-based keystore for which you want to create the auto-login keystore.

   For Example:

   ```
   ADMINISTER KEY MANAGEMENT CREATE LOCAL AUTO_LOGIN KEYSTORE FROM KEYSTORE
   '/etc/ORACLE/WALLETS/orcl' IDENTIFIED BY "asdf1234";
   ```

   After you run this statement, the **cwallet.sso** file appears in the keystore location. The **ewallet.p12** file is the password-based wallet.

   NOTE: Rename the **ewallet.p12** to **ewallet.p24**.

NOTE: Do not remove the PKCS#12 wallet (**ewallet.p12** file) after you create the auto login keystore (**.sso file**). You must have the PKCS#12 wallet to regenerate or rekey the TDE master encryption key in the future.

Migrating Oracle Wallet to HSM

Migrating from a Password-Based Software Wallet to a HSM Wallet

Convert the Software Wallet to Open with the HSM wallet

For the software keystore to open with the hardware keystore, follow the steps below:

- The software keystore must have the same password as the hardware keystore.
- Create an auto-login keystore for the software keystore.

Use the following syntax to set the software keystore password as that of the hardware keystore:

```
ADMINISTER KEY MANAGEMENT ALTER KEYSTORE PASSWORD IDENTIFIED BY
software_keystore_password SET "user_id:password" WITH BACKUP [USING
'backup_identifier'];
```

For Example:

```
ADMINISTER KEY MANAGEMENT ALTER KEYSTORE PASSWORD IDENTIFIED BY "asdf1234" SET
"tdeowner:asdf1234" WITH BACKUP USING 'password_backup';
```

In this specification:

- `software_keystore_password` is the same password that you used when creating the software keystore.
- `user_id:password` is the new software keystore password which is the same as the password of the HSM.
- `WITH BACKUP` creates a backup of the software keystore. 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_enc_key_backup.p12**, with **enc_key_backup** being the backup identifier). Follow the file naming conventions that your operating system uses.

Use the following syntax to create an auto-login keystore for a software keystore:

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

For Example:

```
ADMINISTER KEY MANAGEMENT CREATE LOCAL AUTO_LOGIN KEYSTORE FROM KEYSTORE
'/etc/ORACLE/WALLETS/orcl' IDENTIFIED BY "tdeowner:asdf1234";
```

In this specification:

- `LOCAL` enables you to create a local auto-login software keystore.
- `keystore_location` is the path to the keystore directory location of the keystore that is configured in the `sqlnet.ora` file.

- `software_keystore_password` is the existing password of the configured software keystore.

Configure sqlnet.ora for the Migration of the Password-Based Software Wallet

If you are migrating from a software keystore to a hardware keystore, then use the following format in the sqlnet.ora file:

```
ENCRYPTION_WALLET_LOCATION=
  (SOURCE=(METHOD=HSM)(METHOD_DATA=
    (DIRECTORY=path_to_keystore)))
```

`path_to_software_keystore` is the path to the previously configured software keystore.

For Example:

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

Perform the HSM wallet Migration

After you complete the migration, you do not need to restart the database, nor do you need to manually re-open the hardware keystore. The migration process automatically reloads the keystore keys in memory.

Use the following syntax when you run the `ADMINISTER KEY MANAGEMENT SQL` statement for migration:

```
ADMINISTER KEY MANAGEMENT SET ENCRYPTION KEY IDENTIFIED BY "user_id:password"
MIGRATE USING software_keystore_password [WITH BACKUP [USING 'backup_identifier']];
```

For Example:

```
ADMINISTER KEY MANAGEMENT SET ENCRYPTION KEY IDENTIFIED BY "tdeowner:asdf1234"
MIGRATE USING "tdeowner:asdf1234" WITH BACKUP USING 'migration_backup';
```

In this specification:

- `user_id:password` is the user ID and password that was created on SafeNet KeySecure. Enclose this setting in double quotation marks (" ") and separate user_id and password with a colon (:).
- `software_keystore_password` is the same password that you have changed to in **"Step 1: Convert the Software Keystore to Open with the Hardware Keystore"**.
- `USING` enables you 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_enc_key_backup.p12**, with **enc_key_backup** being the backup identifier). Follow the file naming conventions that your operating system uses.

Migrating from an Auto-login Software Wallet to an Auto-login HSM Wallet

1. Configure `sqlnet.ora` for the Migration of the Auto-login Software Wallet.

   If you are migrating from a software keystore to a hardware keystore, then use the following format in the `sqlnet.ora` file:

   ```
   ENCRYPTION_WALLET_LOCATION=
     (SOURCE=(METHOD=HSM)(METHOD_DATA=
       (DIRECTORY=path_to_keystore)))
   ```

path_to_software_keystore is the path to the previously configured software keystore.

For Example:

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

2. Perform the HSM wallet Migration

   After you complete the migration, you do not need to restart the database, nor do you need to manually re-open the hardware keystore. The migration process automatically reloads the keystore keys in memory.

   Use the following syntax when you run the ADMINISTER KEY MANAGEMENT SQL statement for migration:

   ```
   ADMINISTER KEY MANAGEMENT SET ENCRYPTION KEY IDENTIFIED BY "user_id:password"
   MIGRATE USING software_keystore_password [WITH BACKUP [USING 'backup_identifier']];
   ```

   For Example:

   ```
   ADMINISTER KEY MANAGEMENT SET ENCRYPTION KEY IDENTIFIED BY "tdeowner:asdf1234"
   MIGRATE USING "tdeowner:asdf1234" WITH BACKUP USING 'migration_backup';
   ```

3. If you have migrated and are using an auto-login software keystore in a specific location (for example, /etc/ORACLE/WALLETS/orcl), then create the software password keystore with the hardware keystore password from the auto-login keystore.

   ```
   ADMINISTER KEY MANAGEMENT CREATE KEYSTORE '/etc/ORACLE/WALLETS/HSM' IDENTIFIED BY
   "tdeowner:password";
   ```

   ```
   ADMINISTER KEY MANAGEMENT MERGE KEYSTORE '/etc/ORACLE/WALLETS/orcl' INTO EXISTING
   KEYSTORE '/etc/ORACLE/WALLETS/HSM' IDENTIFIED BY "cdbuser:asdf1234" WITH BACKUP;
   ```

   The location of the keystore for the ADMINISTER KEY MANAGEMENT merge statement does not need to be the location of the keystore in use.

4. Reconfigure the sqlnet.ora file and add the keystore location of the software keystore created in Step 3 to the DIRECTORY setting of the ENCRYPTION_WALLET_LOCATION setting.

   For example:

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

5. Restart the database and create a new sqlplus session.

6. Open the software keystore.

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

   For example:

   ```
   ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN IDENTIFIED BY "tdeowner:asdf1234";
   ```

7. Add or update the secret in the software keystore.

   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.

   ```
   ADMINISTER KEY MANAGEMENT ADD SECRET "user_id:password"
   ```

```
FOR CLIENT "HSM_PASSWORD" IDENTIFIED BY software_keystore_password WITH BACKUP;
```

For example:

```
ADMINISTER KEY MANAGEMENT ADD SECRET 'tdeowner:asdf1234' FOR CLIENT 'HSM_PASSWORD'
IDENTIFIED BY "tdeowner:asdf1234" WITH BACKUP;
```

8. Close the software keystore.

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

For example:

```
ADMINISTER KEY MANAGEMENT SET KEYSTORE CLOSE IDENTIFIED BY "tdeowner:asdf1234";
```

9. Create (or re-create) the auto-login keystore.

```
ADMINISTER KEY MANAGEMENT CREATE AUTO_LOGIN KEYSTORE FROM KEYSTORE
'/etc/ORACLE/WALLETS/HSM' IDENTIFIED BY "tdeowner:asdf1234";
```

10. A new file, **cwallet.sso**, is created at the `/etc/ORACLE/WALLETS/HSM` directory. This directory now contains two files, **cwallet.sso** and **ewallet.p12**.

NOTE: Rename the **ewallet.p12** to **ewallet.p24**.

11. Update the `sqlnet.ora` file to use the hardware security module location.

For example:

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

At this stage, the next time that a TDE operation executes, the hardware security module auto-login keystore opens automatically.

About Migrating Back from a Hardware Keystore

If you want to switch from using a hardware keystore solution to a software keystore, then you can use reverse migration of the keystore.

After you complete the switch, keep the hardware security module, in case earlier backup files rely on the TDE master encryption keys in the hardware security module.

Migrating back from a Password based Hardware Keystore to a Password based Or Auto login Software Keystore

1. Configure `sqlnet.ora` for the Reverse Migration.

First edit the sqlnet.ora file.

Set the following configuration in the sqlnet.ora file:

```
ENCRYPTION_WALLET_LOCATION= (SOURCE= (METHOD=FILE) (METHOD_DATA=
(DIRECTORY=path_to_keystore)))
```

Replace path_to_keystore with the directory location of the destination keystore.

2. Configuring the Keystore for the Reverse Migration

   To perform a reverse migration on a keystore, you can use the ADMINISTER KEY MANAGEMENT statement with the SET ENCRYPTION KEY and REVERSE MIGRATE clauses.

   a. Log in to the database instance as a user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.

      For example, to log in to the root:

      ```
      sqlplus / as sysdba

      connected to: Oracle Database 12c

      SQL> GRANT ADMINISTER KEY MANAGEMENT to system;

      SQL> commit;

      Commit complete.

      SQL> Connect system/<system_password>

      Connected.
      ```

   b. Run the following SQL statement:

      ```
      ADMINISTER KEY MANAGEMENT SET ENCRYPTION KEY IDENTIFIED BY
      software_keystore_password REVERSE MIGRATE USING "user_id: password" [WITH BACKUP
      [USING 'backup_identifier']];
      ```

      In this specification:

   - `software_keystore_password` is the password for the existing keystore or the new keystore.
   - `user_id:password` is the software keystore password which is the same as the password of the HSM
   - `WITH BACKUP` creates a backup of the software keystore. Optionally, you can include 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). Follow the file naming conventions that your operating system uses.

     For example:

     ```
     ADMINISTER KEY MANAGEMENT SET ENCRYPTION KEY IDENTIFIED BY "tdeowner: asdf1234"
     REVERSE MIGRATE USING "tdeowner: asdf1234" WITH BACKUP;

     Keystore altered.
     ```

   c. Optionally, change the keystore password using the following syntax

      ```
      ADMINISTER KEY MANAGEMENT ALTER KEYSTORE PASSWORD IDENTIFIED BY
      software_keystore_password SET "user_id: password" WITH BACKUP [USING
      'backup_identifier'];
      ```

      For Example:

      ```
      ADMINISTER KEY MANAGEMENT ALTER KEYSTORE PASSWORD IDENTIFIED BY "tdeowner:asdf1234"
      SET "asdf1234" WITH BACKUP USING 'password_backup';
      ```

In this specification:

- `software_keystore_password` is the password for the existing keystore or the new keystore.
- `user_id: password` is the software keystore password.
- `WITH BACKUP` creates a backup of the software keystore. Optionally, you can include 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). Follow the file naming conventions that your operating system uses.

3. Configuring the Hardware Keystore to Open with the Software Keystore

   After you complete the migration, you do not need to restart the database, nor do you need to manually re-open the software keystore. The migration process automatically reloads the keystore keys in memory.

   The hardware keystore may still be required after reverse migration because the old keys are likely to have been used for encrypted backups or by tools such as Oracle Data Pump and Oracle Recovery Manager. You should cache the hardware keystore credentials in the keystore so that the HSM can be opened with the software keystore.

Reverse Migration of Password based HSM wallet to Password based oracle wallet

1. Ensure that you configured the TDE hardware keystore.
2. Close the hardware security module if it is open. (You can check the status of whether a keystore is open or closed by querying the STATUS column of the V$ENCRYPTION_WALLET view.)

   For example:

   ```
   ADMINISTER KEY MANAGEMENT SET KEYSTORE CLOSE IDENTIFIED BY "tdeowner: asdf1234";
   ```

3. If you have not migrated from a software keystore, then create the software keystore with the hardware keystore password in the appropriate location (for example, /etc/ORACLE/WALLETS/orcl).

   For example:

   ```
   ADMINISTER KEY MANAGEMENT CREATE KEYSTORE '/etc/ORACLE/WALLETS/orcl' IDENTIFIED BY
   "tdeowner:asdf1234";
   ```

4. If you have migrated and are using an auto-login software keystore in a specific location (for example, /etc/ORACLE/WALLETS/HSM), then create the software password keystore with the hardware keystore password from the existing keystore.

   For example:

   ```
   ADMINISTER KEY MANAGEMENT CREATE KEYSTORE '/etc/ORACLE/WALLETS/orcl' IDENTIFIED BY
   "tdeowner: asdf1234";

   ADMINISTER KEY MANAGEMENT MERGE KEYSTORE '/etc/ORACLE/WALLETS/HSM' INTO EXISTING
   KEYSTORE '/etc/ORACLE/WALLETS/orcl' IDENTIFIED BY "tdeowner:asdf1234" WITH BACKUP;
   ```

5. The location of the keystore for the ADMINISTER KEY MANAGEMENT merge statement does not need to be the location of the keystore in use.
6. Reconfigure the sqlnet.ora file and add the keystore location of the software keystore created in Step 3 or Step 4 to the DIRECTORY setting of the ENCRYPTION_WALLET_LOCATION setting.

   For example:

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

7. Reconnect to the database, or log out and then log back in again, so that the change that you made in the previous step takes effect.

   For example:

   ```
   SQL> shutdown immediate;

   Database closed.

   SQL> startup

   ORACLE instance started.

   sqlplus / as sysdba

   connected to: Oracle Database 12c
   ```

8. Open the software keystore.

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

   For example:

   ```
   ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN IDENTIFIED BY "asdf1234";
   ```

9. Add or update the secret in the software keystore.

   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.

   For example:

   ```
   ADMINISTER KEY MANAGEMENT ADD SECRET "tdeowner:asdf1234” FOR CLIENT "HSM_PASSWORD"
   IDENTIFIED BY "asdf1234" WITH BACKUP;
   ```

10. Re-open the software keystore.

Reverse Migration of Password Based HSM wallet to Auto login oracle wallet

1. Create (or re-create) the auto-login keystore.

   For example:

   ```
   ADMINISTER KEY MANAGEMENT CREATE AUTO_LOGIN KEYSTORE FROM KEYSTORE
   '/etc/ORACLE/WALLETS/orcl/hsm' IDENTIFIED BY "asdf1234";
   ```

2. Restart the database and SQL session.

Migrating back from an Auto Login Hardware Keystore to a Password based Or Auto login Software Keystore

Reverse Migration of Auto login HSM wallet to Auto login oracle wallet

Perform Step 1 and 2 of Migrating back from a Password based Hardware Keystore to a Password based Or Auto login Software Keystore.

Reverse Migration of Auto login HSM wallet to Password based oracle wallet

1. Perform the Step 1 and 2 of Migrating back from a Password based Hardware Keystore to a Password based Or Auto login Software Keystore.

2. Rename the `cwallet.sso` file to `cwallet.back`
3. Reopen the software keystore.

Working with Pluggable Databases (PDB)

A new option for Oracle Database 12c, 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

Once you are done with creating a pluggable database using the DBCA utility, follow the below steps to start working with the newly created PDB:

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 `sqlnet.ora` file.

   For Example:

```
PDB1 =
  (DESCRIPTION =
    (ADDRESS = (PROTOCOL = TCP)(HOST = oracle.localdomain)(PORT = 1521))
    (CONNECT_DATA =
      (SERVER = DEDICATED)
      (SERVICE_NAME = pdb1.localdomain)
```

Where, **PDB1** 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
connected to: Oracle Database 12c
SQL> alter pluggable database all open read write;
```

> **NOTE:** After every Database restart it is required to run the `alter pluggable` command (alter pluggable database all open read write;).

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

For Example:

```
SQL> connect system/temp123#@pdb1
Connected.
```

4. Run the below grant commands:

```
SQL> GRANT ADMINISTER KEY MANAGEMENT TO PDB1USER;
SQL> GRANT CREATE SESSION TO PDB1USER;
SQL> GRANT CONNECT TO PDB1USER;
SQL> COMMIT;
```

Where, **pdb1user** 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
Connected
SQL> connect pdb1user/temp123#@pdb1
Connected.
```

Managing HSM Wallets: PDB

Encryption wallet can be of two types: HSM (SafeNet KeySecure) and Oracle (software) wallets. It is recommended to use an HSM rather than Oracle wallet.

> **NOTE:** This section describes steps to perform fresh configuration of an HSM wallet. The steps described assume that no HSM or Oracle wallet already exists. For information on configuring Oracle wallets, see “Managing Oracle Wallets”.

Configuring HSM Wallets

1. **Set the Hardware Keystore Type in the sqlnet.ora File**
2. **Open the Hardware Keystore in CDB (Container database)**
3. **Set the Hardware Keystore TDE Master Encryption Key in CDB**
4. **Open the Hardware Keystore in PDB (Pluggable database)**
5. **Set the Hardware Keystore TDE Master Encryption Key in PDB**
6. **Encrypt Your Data**
7. **Close the Hardware Keystore**

Set the Hardware Keystore Type in the sqlnet.ora File

Before you can configure a hardware keystore, you must enable the database to recognize that it is a hardware security module by editing the sqlnet.ora file. By default, this file is located in the ORACLE_HOME/network/admin directory or in the location set by the TNS_ADMIN environment variable.

Use the following setting in the sqlnet.ora file to define the hardware keystore type, which is HSM.

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

Open the Hardware Keystore in CDB (Container database)

1. Log in to the database instance as a user who has been granted the ADMINISTER KEY MANAGEMENT or SYSKM privilege.

   For example, to log in to the root:

   ```
   sqlplus / as sysdba
   connected to: Oracle Database 12c
   SQL> GRANT ADMINISTER KEY MANAGEMENT to system;
   Grant succeeded.
   SQL> commit;
   Commit complete.
   SQL> Connect system/<system_password>@cdbDatabase
   Connected.
   ```

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

   ```
   ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN IDENTIFIED BY "user_id:password";
   ```

   For example:

   ```
   ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN IDENTIFIED BY "tdeowner:asdf1234";
   keystore altered.
   ```

Set the Hardware Keystore TDE Master Encryption Key in CDB

1. Run the following SQL statement:

```
ADMINISTER KEY MANAGEMENT SET KEY IDENTIFIED BY "user_id:password";
```

   For example:

```
ADMINISTER KEY MANAGEMENT SET KEY IDENTIFIED BY "tdeowner:asdf1234";

keystore altered.
```

Open the Hardware Keystore in PDB (Pluggable database)

```
Connect pdbuser/<system_password>@pdbDatabase
```

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

```
ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN IDENTIFIED BY "user_id:password";
```

   For example:

```
ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN IDENTIFIED BY "tdeowner:asdf1234";

keystore altered.
```

2. Run the following SQL statement:

```
ADMINISTER KEY MANAGEMENT SET KEY IDENTIFIED BY "user_id:password";
```

   For example:

```
ADMINISTER KEY MANAGEMENT SET KEY IDENTIFIED BY "tdeowner:asdf1234";

keystore altered.
```

   This will generate a new master key and from now onwards any operations performed within this pdb will use the same key.

Managing Oracle Wallets: PDB

Configuring Oracle Wallets

Set the Hardware Keystore Type in the sqlnet.ora File

Before you can configure a hardware keystore, you must enable the database to recognize that it is a hardware security module by editing the sqlnet.ora file. By default, this file is located in the *ORACLE_HOME/network/admin* directory or in the location set by the TNS_ADMIN environment variable.

Use the following setting in the sqlnet.ora file to define the software keystore type, which is FILE.

```
ENCRYPTION_WALLET_LOCATION=

(SOURCE=(METHOD=FILE)(METHOD_DATA=

(DIRECTORY=path_to_keystore)))
```

Open the Hardware Keystore in CDB (Container database)

1. Log in to the database instance as a user who has been granted the ADMINISTER KEY MANAGEMENT or SYSKM privilege.

   For example, to log in to the root:

```
sqlplus / as sysdba
connected to: Oracle Database 12c
SQL> GRANT ADMINISTER KEY MANAGEMENT to system;
Grant succeeded.
SQL> commit;
Commit complete.
SQL> Connect system/<system_password>@cdbDatabase
Connected.
```

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

```
ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN IDENTIFIED BY "password";
```

For example:

```
ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN IDENTIFIED BY "asdf1234";
keystore altered.
```

Set the Hardware Keystore TDE Master Encryption Key in CDB

1. Run the following SQL statement:

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

For example:

```
ADMINISTER KEY MANAGEMENT SET KEY IDENTIFIED BY "asdf1234";
keystore altered.
```

Open the Hardware Keystore in PDB (Pluggable database)

```
Connect pdbuser/<system_password>@pdbDatabase
```

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

```
ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN IDENTIFIED BY "password";
```

For example:

```
ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN IDENTIFIED BY "asdf1234";
keystore altered.
```

2. Run the following SQL statement:

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

For example:

```
ADMINISTER KEY MANAGEMENT SET KEY IDENTIFIED BY "asdf1234";
keystore altered.
```

This will generate a new master key and from now onwards any operations performed within this pdb will use the same key.

NOTE: To enable the auto login in PDB, we first need to create the auto login wallet in container database and this will work for pluggable database as well. We do not require further steps to perform in PDB.

Migrating Oracle Wallets to HSM: PDB

To migrate Oracle Wallets to HSM, follow the steps below:

1. Perform migration to HSM.

   Change the wallet password same as HSM.

   ```
   ADMINISTER KEY MANAGEMENT ALTER KEYSTORE PASSWORD IDENTIFIED BY
   software_keystore_password SET "user_id:password" WITH BACKUP [USING
   'backup_identifier'];
   ```

   For Example:

   ```
   ADMINISTER KEY MANAGEMENT ALTER KEYSTORE PASSWORD IDENTIFIED BY "asdf1234" SET
   "tdeowner:asdf1234" WITH BACKUP USING 'password_backup';
   ```

In this specification:

- `software_keystore_password` is the same password that you used when creating the software keystore.
- `user_id:password` is the new software keystore password which is the same as the password of the HSM.

`WITH BACKUP` creates a backup of the software keystore. 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_enc_key_backup.p12**, with **enc_key_backup** being the backup identifier). Follow the file naming conventions that your operating system uses.

2. Change the method to HSM in `sqlnet.ora` file and then perform migration.

   ```
   ENCRYPTION_WALLET_LOCATION = (SOURCE = (METHOD = HSM) (METHOD_DATA = (DIRECTORY =
   'wallet_location')))
   ```

   For Example:

   ```
   ENCRYPTION_WALLET_LOCATION=

     (SOURCE=(METHOD=HSM)(METHOD_DATA=

       (DIRECTORY=/etc/ORACLE/WALLETS/orcl)))
   ```

3. Set the encryption key.

   ```
   ADMINISTER KEY MANAGEMENT SET ENCRYPTION KEY IDENTIFIED BY "user_id:password"
   MIGRATE USING software_keystore_password [WITH BACKUP [USING 'backup_identifier']];
   ```

   For Example:

   ```
   ADMINISTER KEY MANAGEMENT SET ENCRYPTION KEY IDENTIFIED BY "tdeowner:asdf1234"
   MIGRATE USING "tdeowner:asdf1234" WITH BACKUP USING 'migration_backup';
   ```

4. Now restart the database, login with CDB and open the wallet.
5. Login in Pluggable database and open the wallet. HSM wallet will open and retrieve data from encrypted tables.

Plugging an Unplugged Pluggable Database

The CDBs keystore is used to store encryption keys for all the associated PDBs. The master encryption key for the PDB must be exported before an unplug operation, so it can be imported after a subsequent plugin operation.

You can disassociate or unplug a PDB from a CDB and re-associate or plug the PDB into the same CDB or into another CDB.

This section describes the process of unplugging PDB1 from the CDB1 instance and plugging into the CDB2 instance on the same machine with a name of PDB1.

1. **Export the key information from PDB**
2. **Unplug the PDB**
3. **Plugin the PDB**
4. **Import the key information of PDB**

Preparations:

a. Two CDB's (CDB1 and CDB2) are configured
b. PDB1 is attached to CDB1.
c. Master encryption key for CDB1, CDB2 and PDB1 is set.

Switch to the CDB1 instance

- ORAENV_ASK=NO
- export ORACLE_SID=cdb1
- . oraenv
- ORAENV_ASK=YES

Export the key information from PDB

1. Restart the Listener Service
   - lsnrctl stop
   - lsnrctl start
2. Start the sqlplus session to connect to PDB.

```
sqlplus / as sysdba

connected to: Oracle Database 12c

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

NOTE: After every DB restart it is required to run the alter pluggable command (alter pluggable database all open read write;).

3. Connect to the PDB

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

For Example:

```
SQL> connect system/temp123#@pdb1
Connected.
```

4. Export the PDB keys

```
ADMINISTER KEY MANAGEMENT EXPORT ENCRYPTION KEYS WITH
SECRET "secret" TO 'key_file' IDENTIFIED BY software_keystore_password;
```

where,

- `software_keystore_password` is the keystore password.
- `secret` is the password used to encrypt the exported key.
- `Key_file` is the file to which encrypted key information is stored.

Example:

```
ADMINISTER KEY MANAGEMENT EXPORT ENCRYPTION KEYS WITH
SECRET "secret" TO '/tmp/export.p12' IDENTIFIED BY
"asdf1234";
```

Unplug the PDB

To unplug a PDB, you first close it and then generate an XML manifest file. The XML file contains information about the names and the full paths of the tablespaces, as well as data files of the unplugged PDB. The information will be used by the plugging operation.

1. Start the sqlplus session to connect to PDB.

```
sqlplus / as sysdba;
connected to: Oracle Database 12c
```

2. Close the PDB

```
alter pluggable database <PDB NAME> close immediate;
```

Example:

```
alter pluggable database pdb1 close immediate;
```

3. Unplug the closed PDB and then specify the path and name of the XML file.

```
alter pluggable database <PDB NAME> unplug into 'XML_FILE_PATH’;
<XML_FILE_PAHT> is the manifest file that contains information about the names and
the full paths of the tablespaces, as well as data files of the unplugged PDB.
```

Example:

```
alter pluggable database pdb1 unplug into '/home/oracle/pdb1.xml';
```

4. Drop the closed PDB and keep the data files.

```
drop pluggable database <PDB_NAME>  keep datafiles;
```

Example:

```
drop pluggable database pdb1 keep datafiles;
```

5. Verify the status of the unplugged PDB.

```
select pdb_name, status from cdb_pdbs where pdb_name in ('PDB1');
```

No rows are seen.

The unplugging operation makes changes in the PDB data files to record that the PDB was properly and successfully unplugged.

NOTE: Unplug procedures can also be done through Database Configuration Assistant.

Procedure:

1. Set the container database on oracle environment.
2. If pdb has to be removed from CDB1, give CDB1.

Example:

```
. oraenv

ORACLE_SID = [cdb2]? cdb1
```

3. Go to the path shown by echo $ORACLE_HOME
4. Enter dbca and follow the options.

Plugin the PDB

1. Checking the Compatibility of the Unplugged PDB with the Host CDB

Before starting the plugging operation, make sure that the to-be-plugged-in PDB is compatible with the new host CDB. Execution of the PL/SQL block raises an error if it is not compatible.

Execute the following PL/SQL block:

```
. oraenv

[enter cdb2 at the prompt]

sqlplus / as sysdba

[if cdb2 is not started up, start it up now.]

set serveroutput on

DECLARE
   compatible BOOLEAN := FALSE;
BEGIN
   compatible := DBMS_PDB.CHECK_PLUG_COMPATIBILITY(
```

```
        pdb_descr_file => '<XML_FILE_PATH>');
   if compatible then
      DBMS_OUTPUT.PUT_LINE('Is pluggable PDB1 compatible? YES');
  else DBMS_OUTPUT.PUT_LINE('Is pluggable PDB1 compatible? NO');
   end if;
END;
/
```

For example:

```
SQL> set serveroutput on
DECLARE
   compatible BOOLEAN := FALSE;
BEGIN
   compatible := DBMS_PDB.CHECK_PLUG_COMPATIBILITY(
        pdb_descr_file => '/home/oracle/pdb1.xml');
   if compatible then
      DBMS_OUTPUT.PUT_LINE('Is pluggable PDB1 compatible? YES');
   else DBMS_OUTPUT.PUT_LINE('Is pluggable PDB1 compatible? NO');
   end if;
END;
/
Is pluggable PDB1 compatible? YES
PL/SQL procedure successfully completed.
```

<XML_FILE_PAHT> is the manifest file that contains information about the names and the full paths of the tablespaces, as well as data files of the unplugged PDB.

2. Plugging the Unplugged PDB: NOCOPY Method

```
create pluggable database <PDB NAME> using 'XML_FILE_PATH' NOCOPY TEMPFILE REUSE;
```

<XML_FILE_PAHT> is the manifest file that contains information about the names and the full paths of the tablespaces, as well as data files of the unplugged PDB.

Example:

```
create pluggable database pdb1 using '/home/oracle/pdb1.xml' NOCOPY TEMPFILE REUSE;
```

NOTE:

- There are two more methods which can be used for plugging the PDB. Please refer the http://www.oracle.com/webfolder/technetwork/tutorials/obe/db/12c/r1/pdb/pdb_unplug_plug/pdb_unplug_plug.html

  OR

- You can use dbca utility to do this. Please refer https://oracle-base.com/articles/12c/multitenant-create-and-configure-pluggable-database-12cr1.

Import the key information of PDB

1. Import the PDB keys

```
ADMINISTER KEY MANAGEMENT IMPORT ENCRYPTION KEYS WITH SECRET "secret" FROM
'key_file' IDENTIFIED BY "software_keystore_password" WITH BACKUP;
```

   Example:

```
ADMINISTER KEY MANAGEMENT IMPORT ENCRYPTION KEYS WITH SECRET "secret" FROM
'/tmp/export.p12' IDENTIFIED BY "asdf1234" WITH BACKUP;
```

2. Restart the Database and access the PDB.

11

Integrating TDE with SafeNet KeySecure on Oracle 12c RAC (12.1.0.2 – 64 bit)

This chapter contains the following information:

Setting up SafeNet KeySecure for Transparent Data Encryption (TDE) with Oracle RAC

Managing Multiple Databases on RAC

Setting up SafeNet KeySecure for Transparent Data Encryption (TDE) 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. There are several ways to check the status of the RAC. The *srvctl* utility shows the current configuration and status of the RAC database.

```
$ . oraenv

ORACLE_SID = [ORCL] ? ORCL

The Oracle base remains unchanged with value /u01/app/oracle

[oracle@rac1 ~]$ srvctl config database -d ORCL

Database unique name: ORCL

Database name: ORCL

Oracle home: /u01/app/oracle/product/12.1.0.2/db_1

Oracle user: oracle

Sp_file: +DATA/ORCL/PARAMETERFILE/spfile.369.912725063

Password file: +DATA/ORCL/PASSWORD/pwdORCL.347.912724045

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:
Database instances: ORCL1,ORCL2
Configured nodes: rac1,rac2
Database is administrator managed
$ srvctl status database -d ORCL
Instance ORCL1 is running on node rac1
Instance ORCL2 is running on node rac2

The V$ACTIVE_INSTANCES view can also display the current status of the instances.
$ export ORACLE_SID=ORCL1
[oracle@rac1 Desktop]$ sqlplus / as sysdba
SELECT inst_name FROM v$active_instances;
INST_NAME
---------------------------------------------------------------------------
rac1.localdomain:ORCL1
rac2.localdomain:ORCL2

exit
$
```

NOTE: The above example shows 2x1 architecture, where 2 is the number of nodes (rac1 and rac2) and 1 is the number of databases (orcl).

However; our library supports more complex architecture where, the number of nodes and databases may be higher, for example 3x3 i.e. 3 nodes (rac1, rac2 and rac3) and 3 databases (orclA, orclB and orclC).

Configuring the PKCS11 Provider on Oracle RAC Instances

Configuration of SafeNet PKCS#11 with Oracle TDE on RAC involves the following steps:

1. **Installing the Application Server License on SafeNet KeySecure**
2. **Creating Oracle TDE Authentication Credentials**
3. **Installing SafeNet PKCS#11 Library on Linux (to be performed on both RAC1 and RAC2)**
4. **Configuring Connection to SafeNet KeySecure (to be performed on both RAC1 and RAC2)**

Installing the Application Server License on SafeNet KeySecure

To install the Application Server License on SafeNet KeySecure:

1. Obtain an Application Server License file from SafeNet.
2. Install the license file on the SafeNet KeySecure.

   To install the license file:

   a. Log on to the **Management Console** as an administrator.

   b. Navigate to the System Information page (**Device >> System Information & Upgrade**).

   c. Select the method of installation and click **Upgrade/Install**. The machine will reboot.

Creating Oracle TDE Authentication Credentials

The Oracle database needs a user to authenticate to the SafeNet KeySecure. This user will own the keys generated by Oracle TDE.

To create Oracle TDE authentication credentials on SafeNet KeySecure:

1. Log on to the Management Console as an administrator with Users and Groups access control. (The `admin` account created during the SafeNet KeySecure installation has this access control.)
2. Navigate to the Local Users section of the User & Group Configuration page (**Security >> Local Authentication >> Local Users & Groups**).
3. Click **Add**.
4. Enter a user name in the **Username** field and password in the **Password** field. (This document assumes **tdeowner** as the user name and **asdf1234** as password.)s
5. Select the **User Administration Permission** and **Change Password Permission** check boxes.
6. Click **Save**.

Installing SafeNet PKCS#11 Library on Linux (to be performed on RAC1 and RAC2)

To install SafeNet PKCS#11 library:

1. Download SafeNet PKCS#11 from SafeNet customer support site: https://serviceportal.safenet-inc.com. The software adheres to the following naming convention:

   ```
   SafeNet Part Number - Product Name - Product Version - File Format
   ```

For example:

```
610-013046-001_pkcs11_tde_linux_64b_v8.3.0.000-0xx.tar.gz
```

2. Log on to the Linux/UNIX client as Oracle user.
3. Extract the file using any standard archive utility.

   For example, execute:

   ```
   tar -xzf <source_directory/tar_file_name> -C <destination_directory>
   ```

4. Create the `/opt/oracle/extapi/<ARCH>/hsm/safenet/<VERSION>` directory. The Oracle user must have appropriate access permissions on /opt/.

   Where `<ARCH>` is the system architecture (either 32 or 64), and `<VERSION>` is the SafeNet software version number (e.g., 8.3.0).

   From this point onward, this document uses `<ARCH>` as **64** and `<VERSION>` as 8.3.0. If the system architecture and version is different, adjust these values accordingly.

5. Copy the library file `libIngPKCS11.so-8.3.0.000` from extracted `/SafeNet/PKCS11/lib` directory to `/opt/oracle/extapi/64/hsm/safenet/8.3.0`.

   For example:

   ```
   $ cp libIngPKCS11.so-8.3.0.000 /opt/oracle/extapi/64/hsm/safenet/8.3.0
   ```

   NOTE: The receiving directory is a fixed location. Oracle searches this directory. It cannot be changed. Changing the directory name results in a "`cannot find PKCS11 library`" error.

6. Copy the properties file `IngrianNAE.properties` from extracted `/SafeNet/PKCS11` directory to `/opt/oracle/extapi/64/hsm/safenet/8.3.0`.

   For example:

   ```
   $ cp IngrianNAE.properties /opt/oracle/extapi/64/hsm/safenet/8.3.0
   ```

7. Rename **libIngPKCS11.so-8.3.0.000** as **libIngPKCS11.so**.

   For example:

   ```
   $ mv libIngPKCS11.so-8.3.0.000 libIngPKCS11.so
   ```

Configuring Connection to SafeNet KeySecure (to be performed on RAC1 and RAC2)

To configure connection to SafeNet KeySecure:

1. Configure the connection to SafeNet KeySecure.

   Enter the following values in the IngrianNAE.properties file (placed at `/opt/oracle/extapi/64/hsm/safenet/8.3.0`).

   - **NAE_IP** – IP address of the SafeNet KeySecure.
   - **NAE_Port** – 9000 (This is the default value).
   - **Log_Level** – MEDIUM (This is the default, but it can be set to HIGH for troubleshooting).

- **Log_File** – Full path and file name. The Oracle user must have write permissions for the path and file. A public location such as `/tmp` is recommended.

2. Modify environment variables for the Oracle user.

   Make sure that the following environment variables are exported so that they are inherited by new Oracle server processes. Edit the shell profile (in many shells, the file is called **.profile** and is located in the home directory of the Oracle user.)

   ```
   export SFNT_HSMAPI_BASE=/opt/oracle/extapi/<32|64>/hsm/safenet/8.3.0

   export NAE_Properties_Conf_Filename=$SFNT_HSMAPI_BASE/IngrianNAE.properties

   export IngrianNAE_Properties_Conf_Slot_ID_Max=100

   export IngrianNAE_Properties_Conf_SessionID_Max=100
   ```

   and run the following commands:

   ```
   srvctl setenv database –d ORCL  -T
   "NAE_Properties_Conf_Filename=/opt/oracle/extapi/64/hsm/safenet/8.3.0/IngrianNAE.pr
   operties"

   srvctl setenv database –d ORCL –T "IngrianNAE_Properties_Conf_SessionID_Max=100"

   srvctl setenv database –d ORCL –T

   "IngrianNAE_Properties_Conf_Slot_ID_Max=100"

   srvctl setenv database –d ORCL –T "ORACLE_UNQNAME=ORCL"
   ```

   where,

   `ORCL` is the name of the database.

   NOTE: If user wants to use multiple properties file for multiple Oracle homes, then environment variables need to be modified as per each Oracle home by using above commands.

3. Add the environment variable (AIX and Solaris).

   - **AIX** – Add the environment variable, LIBPATH, as follows:

     ```
     export LIBPATH=/opt/oracle/extapi/64/hsm/safenet/8.3.0:
     /home/oracle/SafeNet/PKCS11/samplelibs
     ```

   - **Solaris** – Add the environment variable, LD_LIBRARY_PATH, as follows:

     ```
     export LD_LIBRARY_PATH=/opt/oracle/extapi/64/hsm/safenet/8.3.0:
     /home/oracle/SafeNet/PKCS11/samplelibs
     ```

Here, samplelibs is provided in the SafeNet PKCS#11 TDE package.

Managing Oracle Wallets

Encryption wallet can be of two types: HSM (SafeNet KeySecure) and Oracle (software) wallets. This section describes instructions to configure an Oracle wallet.

Configuring an Oracle Wallet

After configuring SafeNet PKCS#11 with Oracle TDE, the encryption key for TDE can be generated and stored in the Oracle wallet (i.e., in database). Generation of the encryption key involves the following steps:

1. **Adding the Wallet Location to sqlnet.ora**
2. **Restarting the Oracle Database**
3. **Creating the Encryption Wallet**
4. **Encrypting a Column in a Table**
5. **Opening the Wallet Manually**
6. **Checking the Wallet Status**
7. **Closing the Wallet**

Adding the Wallet Location to sqlnet.ora

a. 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 use the shared disk for storing the wallet. If shared disk is used you will not have to copy 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.

b. Create or add the following to `$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)))
```

Restarting the Oracle Database

After making any changes to the **sqlnet.ora** file, it is necessary to restart the database for the changes to be effective. To restart the database, execute the following commands:

```
$ sqlplus / as sysdba
SQL> connect / as sysdba
Connected
SQL> shutdown immediate
```

```
Database closed.
Database dismounted.
ORACLE instance shut down.
SQL> startup
ORACLE instance started.
Total System Global Area 167772160 bytes
Fixed Size 1247900 bytes
Variable Size 75498852 bytes
Database Buffers 88080384 bytes
Redo Buffers 2945024 bytes
Database mounted.
Database opened.
SQL> exit
$ lsnrctl stop
$ lsnrctl start
```

Creating the Encryption Wallet

After restarting the database, the encryption wallet can be created to store the encryption keys.

In Oracle 12c, a unified master encryption key is created. This key can be used for both TDE column and TDE tablespace encryption. The master encryption key can be created, stored, and rotated in the Oracle wallet.

To create an encryption wallet (at the location specified in the sqlnet.ora file) and add a master encryption key to it, execute the following commands:

1. `$ sqlplus system/<system_password>`

   At this point, no Oracle wallet exists. This can be confirmed by executing:

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

   The command output will display `WRL_TYPE` (wallet type) as `file` and `STATUS` as `CLOSED`.

2. Create a wallet at the location specified in the sqlnet.ora file

   ```
   SQL> ADMINISTER KEY MANAGEMENT CREATE KEYSTORE '/etc/oracle/wallet/RAC'
   IDENTIFIED BY "asdf1234";
   ```

3. After creating the wallet, open the wallet:

   ```
   SQL> ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN IDENTIFIED BY "asdf1234"
   ```

4. `SQL> ADMINISTER KEY MANAGEMENT SET KEY identified by "asdf1234" WITH BACKUP USING 'backup_identifier';`

   Copy the **ewallet.p12** file created in the directory `/etc/oracle/wallet/RAC` from RAC1 to RAC2 in the same directory on RAC2.

NOTE:

- In this sample command, "asdf1234" represents the password to access the Oracle wallet. The password is case-sensitive and must appear in double-quotes (").
- As the encryption key is stored in Oracle wallet, the command requires only the password, not the NAE user name. However, in case of HSM wallet, where the encryption key is created and stored on HSM (SafeNet KeySecure), the command requires both the NAE user name (the key owner) and its password.
- The above command creates the ewallet.p12 file at /etc/oracle/wallet/RAC.

5. Confirm the wallet status. Execute the following command:

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

The command output will display WRL_TYPE (wallet type) as file and STATUS as OPEN. This confirms that the Oracle wallet has been created.

NOTE: Re-executing the command "ADMINISTER KEY MANAGEMENT SET KEY identified by "asdf1234";" rotates (re-keys) the master encryption key. Every time this command is executed, a new encryption key is generated and stored in Oracle wallet, and the data is encrypted with the new encryption key.

Encrypting a Column in a Table

To encrypt a column in a table, execute the following command:

```
SQL> alter table <table_owner>.<table_name> modify (<column_name> encrypt);
```

Here <table_owner> represents the name of the table owner and <table_name> represents the name of the table containing the column <column_name> to be encrypted.

For example:

```
SQL> alter table appowner.credit_cards modify (ccnum encrypt);
```

To verify whether the encrypted data is accessible, execute the following command:

```
SQL> select * from <table_owner>.<table_name>;
```

For example:

```
SQL> select * from appowner.credit_cards;
```

Opening the Wallet Manually

To open the wallet, execute:

```
SQL> ADMINISTER KEY MANAGEMENT SET KEYSTORE open identified by "asdf1234";
```

Checking the Wallet Status

To check the wallet status, execute:

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

Closing the Wallet

After encrypting the column, close the wallet by executing:

```
SQL> ADMINISTER KEY MANAGEMENT SET KEYSTORE CLOSE identified by "asdf1234";
SQL> exit;
```

Above commands work for both RAC1 and RAC2, after copying the wallet on both instances.

Enabling Auto Login with Oracle Wallet

These steps assume that Oracle TDE is used only in "**FILE**" mode. To enable auto login with Oracle wallet:

1. Make sure that the wallet location in the sqlnet.ora file –on both instances of RAC (for e.g. RAC1, RAC2) is:

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

   Restart the database, if needed
2. Make sure that the wallet is open.

   To check the wallet status, execute:

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

   If the wallet is closed, open the wallet by executing: alter system set wallet open identified by `"asdf1234";`
3. Navigate to `/etc/oracle/wallet/RAC` directory.
4. Create a (local) auto-open encryption wallet at `/etc/oracle/wallet/RAC`. Execute the following command:

   ```
   $ orapki wallet create -wallet . -auto_login
   ```

   Here `.` represents the current directory, which should be `/etc/oracle/wallet/RAC`. Enter this path if the current directory is not `/etc/oracle/wallet/RAC`.
5. Enter the wallet password (`asdf1234`) when prompted.
6. A new file, **cwallet.sso**, is created at the `/etc/oracle/wallet/RAC` directory. This directory now contains two files, **cwallet.sso** and **ewallet.p12**.

   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.
7. Close the Oracle wallet. Execute the following command:

   ```
   SQL> ADMINISTER KEY MANAGEMENT SET KEYSTORE close identified by "asdf1234";
   ```

8. Open the wallet again. Run the following command:

```
SQL> ADMINISTER KEY MANAGEMENT SET KEYSTORE open identified by "asdf1234";
```

9. Validate auto login.

   Restart the database, and query a table in an encrypted tablespace. If the query succeeds, then auto-open has been enabled (because the wallet was automatically opened by Oracle without supplying the password).

From now onward, no password is required to access the data encrypted with the master key.

Migrating Oracle Wallet to HSM

An existing Oracle wallet can be migrated to HSM. After the wallet is migrated, auto login can also be enabled with the wallet migrated to HSM.

Migrating an Oracle wallet to HSM and enabling auto login with the migrated wallet involves the following steps:

1. **Migrating an Oracle Wallet to HSM**
2. **Configuring Auto Login with Migrated Wallet**

Migrating an Oracle Wallet to HSM

1. Use the following syntax to set the software keystore password as that of the hardware keystore:

```
ADMINISTER KEY MANAGEMENT ALTER KEYSTORE PASSWORD IDENTIFIED BY
software_keystore_password SET "user_id:password" WITH BACKUP [USING
'backup_identifier'];
```

   For Example:

```
ADMINISTER KEY MANAGEMENT ALTER KEYSTORE PASSWORD IDENTIFIED BY "asdf1234" SET
"tdeowner:asdf1234" WITH BACKUP USING 'password_backup';
```

   In this specification

   - `software_keystore_password` is the same password, used when creating the software keystore.
   - `user_id:password` is the new software keystore password, same as the password of the HSM.
   - `WITH BACKUP` creates a backup of the software keystore. 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_enc_key_backup.p12**, with **enc_key_backup** being the backup identifier). Follow the file naming conventions as per the operating system.

2. Change the `FILE` to `HSM` in `$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)))
```

3. Restart the database.
4. Execute the following command:

```
SQL> ADMINISTER KEY MANAGEMENT SET KEY identified by "tdeowner:asdf1234" migrate
using "asdf1234";
```

Here "tdeowner:asdf1234" represents the NAE user name and its password and "asdf1234" represents the password to access the existing Oracle wallet.

The Oracle wallet is now successfully migrated to HSM and a master encryption key is generated on SafeNet KeySecure.

5. Check the wallet status by executing:

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

The command output will display entries for file and HSM wallets each with the STATUS as OPEN.

6. Verify whether the data encrypted with Oracle wallet is accessible over HSM.

Execute the following command:

```
SQL> select * from <table_owner>.<table_name>;
```

For example:

```
SQL> select * from appowner.credit_cards;
```

7. Once the encryption key is migrated to SafeNet KeySecure, encryption requests similar to the following appear in the activity logs on SafeNet KeySecure every three seconds.

```
[2012-08-07 17:12:01] INFO 192.168.1.22 [-] tdeowner 100003 Crypto
ORACLE.TDE.HSM.MK.06AF95FC1B8BFD4FA0BFFB6D9D68B36AE3 [op#1 ENCRYPT
AES/CBC/PKCS5Padding] - [Success] [-]
```

These encryptions are a result of Oracle's heartbeat functionality related to TDE, specifically, related to external HSMs.

NOTE: After migration of an auto login Oracle wallet to HSM, the auto login functionality is no longer available with the migrated wallet. Auto login needs to be reconfigured with the migrated wallet.

Configuring Auto Login with Migrated Wallet

To configure auto login with wallet migrated to HSM (SafeNet KeySecure):

1. Make sure that the wallet location in the sqlnet.ora file –on both instances of RAC (for e.g. RAC1, RAC2) is:

```
ENCRYPTION_WALLET_LOCATION =

(SOURCE = (METHOD = HSM)

(METHOD_DATA = (DIRECTORY =/etc/oracle/wallet/RAC)))
```

Restart the database, if needed.

2. Navigate to /etc/oracle/wallet/RAC directory.
3. Enable auto-open HSM. Add the following entry to the empty wallet by executing:

```
$ mkstore -wrl . -createEntry ORACLE.TDE.HSM.AUTOLOGIN any-non-empty-string
```

Enter the wallet password (asdf1234) when prompted.

4. 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;

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.

5. Close the wallet (connection to the HSM). Execute the following command:

```
SQL> ADMINISTER KEY MANAGEMENT SET KEYSTORE CLOSE identified by
"tdeowner:asdf1234";
```

NOTE: If the above command results in the error "`ORA-28365: wallet is not open`", it means the wallet is already closed.

6. Open the wallet again. Run the following command:

```
SQL> ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN identified by "tdeowner:asdf1234";
```

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.

7. Validate auto login.

   Restart the database, and query a table in an encrypted tablespace. If the query succeeds, then auto-open has been enabled (because the wallet was automatically opened by Oracle).

   From now onward, no password is required to access the encrypted data with the master encryption key.

Managing HSM Wallets

Encryption wallet can be of two types: HSM (SafeNet KeySecure) and Oracle (software) wallets. This section describes instructions to configure an HSM wallet.

Configuring HSM Wallets

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.

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

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

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>
```

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

```
SQL> alter system set encryption key identified by "tdeowner:asdf1234";
```

NOTE:

- In this sample command, `"tdeowner:asdf1234"` represents the NAE user name and its password. The NAE user name and password are case-sensitive. They must appear in double-quotes (") and be separated by a colon (:).
- The NAE user specified in the above command is the owner of the encryption key created and stored on SafeNet KeySecure.
- The Key and Policy Configuration page of SafeNet KeySecure Management Console displays the generated master encryption key.

Encrypting a Column in a Table

After the master encryption key is generated, it can be used to encrypt a column of a table or tablespace. To encrypt a column in a table, execute the following command:

```
SQL> alter table <table_owner>.<table_name> modify (<column_name> encrypt);
```

Here `<table_owner>` represents the name of the table owner and `<table_name>` represents the name of the table containing the column `<column_name>` to be encrypted.

For example:

```
SQL> alter table appowner.credit_cards modify (ccnum encrypt);
```

To verify whether the encrypted data is accessible, execute the following command:

```
SQL> select * from <table_owner>.<table_name>;
```

For example:

```
SQL> select * from appowner.credit_cards;
```

NOTE: On Windows platforms, while performing select operations on the encrypted table, the following error may occur: `ORA-28353: failed to open wallet`. This error occurs because the wallet gets automatically closed after exiting a SQL session. Therefore, the wallet must be opened manually before performing any operations on the encrypted table.

Opening the Wallet Manually

To open the wallet, execute:

```
ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN IDENTIFIED BY "tdeowner:asdf1234";
```

Checking the Wallet Status

To check the wallet status, execute:

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

Oracle TDE/PKCS#11 can also be used to perform tablespace encryption.

Closing the Wallet

After encrypting a column in a table, close the wallet by executing:

```
SQL> ADMINISTER KEY MANAGEMENT SET KEYSTORE CLOSE IDENTIFIED BY
"tdeowner:asdf1234";

SQL> exit;
```

IMPORTANT: Oracle TDE/PKCS#11 can also be used to enable auto login wallets and to perform tablespace encryption. For details, see “Enabling Auto Login with HSM” on page 65.

Enabling Auto Login with HSM

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

1. The current entry in sqlnet.ora – file on both instances of RAC (for e.g. RAC1, RAC2):

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

   needs to be changed to:

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

2. 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.

3. 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.

4. 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.

5. Close the connection to the HSM with

```
SQL> ADMINISTER KEY MANAGEMENT SET KEYSTORE close identified by
"tdeowner:asdf1234";
```

And open it one last time with

```
SQL> ADMINISTER KEY MANAGEMENT SET KEYSTORE open identified by "tdeowner:asdf1234";
```

6. Validate auto login.

Restart the database, and query a table in an encrypted tablespace. If the query succeeds, then auto-open has been enabled (because the wallet was automatically opened by Oracle without supplying the password).

From now onwards, no password is required to access the data encrypted with the master key.

TDE Tablespace Encryption

First, open the wallet on RAC1 machine.

```
SQL> ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN identified by "tdeowner:asdf1234";
```

Create an encrypted tablespace in the shared disk.

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

Create a table in the tablespace.

```
SQL>create table customer_payment ( first_name varchar2(11),last_name varchar2(11))
TABLESPACE mytablespace;
```

Insert some values in customer_payment table

```
SQL>insert into customer_payment values('Test','Test');

SQL> commit;
```

Close the wallet

```
SQL> ADMINISTER KEY MANAGEMENT SET KEYSTORE CLOSE identified by
"tdeowner:asdf1234";
```

Now try to access the customer_payment table.

```
SQL> select * from customer_payment;
```

You will get the following error as the wallet is not open:

```
ORA-28365: wallet is not open
```

Now you can go on RAC 2 machine. Firstly open the wallet on RAC2 machine

```
SQL> ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN identified by "tdeowner:asdf1234";
```

Now try to access the customer_payment table

```
SQL> select * from customer_payment;
```

1 row will be listed as entered above in the table.

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.

Managing Multiple Databases on RAC

In case you need to have multiple databases on RAC, then simply create a RAC database using dbca utility and follow the same set of instructions as mentioned above.

You need to make sure of the following while working with the new database:

1. *ORACLE_SID's* on both RAC1 & RAC2 are set to new db instance SID respectively, for e.g. if we have two RAC databases *orcl* and *orclnew* then while working with *orclnew* the *ORACLE _SID* must be set to *orclnew1* on RAC1 and *orclnew2* on RAC2. Similarly for *orcl, ORACLE_SID* must be set to *orcl1* on RAC1 and *orcl2* on RAC2.
2. Create separate wallet locations for both the RAC databases, for e.g. for *orcl* database the wallet can be created at **/home/oracle/orcl/WALLET** and for *orclnew* database the wallet should be created at **/home/oracle/orclnew/WALLET**.
3. All the databases, e.g. *orcl* and *orclnew*, need to share the same properties file for configuration purpose.
4. Use of different NAE users is suggested for different databases. All the NAE users should be covered under one group that has privileges to use the TSE key. This is needed as oracle sends a keyinfo request for TSE key.
5. For any new database make sure to change the db name while running the *export* and *srvctl* commands.

12
Troubleshooting & Tips

This chapter provides information on how to handle problems that occur while working with SafeNet PKCS11/Oracle TDE. This chapter also provides Oracle TDE Platform Matrix, which lists issues occurring on different platforms with different Oracle versions and their workaround, wherever available.

Troubleshooting

This table provides information on problems occurring while working with SafeNet PKCS11/Oracle TDE and how to handle them:

#	Description
1.	**"mkdir: cannot create directory '<directory>': Permission denied"** While creating the `/opt/oracle/extapi/<32|64>/hsm/safenet/8.3.0 directory`, if the "`Permission denied`" error occurs, grant appropriate access permissions on this path to the Oracle user.
2.	**"ORA-28407: Hardware Security Module error detected"** While encrypting data or accessing the encrypted data, if ORA - 28407 occurs, there might exist some issues: in configuration/installation process of PKCS#11 library or with the connectivity to the SafeNet KeySecure appliance or in the IngrianNAE.properties file parameters. Refer the PKCS library log file for exact cause of issue. If Oracle traces show the error **"HSM error trace for ORA-28407. kzthsminit failed in C_Initialize with PKCS 11 error code 5"** then copy IngrianNAE.properties file to $ORACLE_HOME/dbs location and rename this file to NAE.properties.
3.	**ORA-28407: Hardware Security Module failed with PKCS#11 error CKR_SLOT_ID_INVALID(%d)** For Oracle 12c RAC auto login with HSM, if error ORA-28407 occurs with invalid slot id then the user must upgrade their Oracle to 12.1.0.2.2 or later.
4.	**Error: "ORA-28365: wallet is not open"** While closing the wallet by executing `"alter system set encryption wallet close identified by "<NAE user>:<password>";"`, if the above error occurs: It means the wallet is already closed. No action is required.
5.	**Linux/Unix: "ORA-28376: cannot find PKCS11 library"** Ensure that `oracle:oinstall` is the `owner:group` of the directory

	'/opt/oracle/extapi/<32|64>/hsm/safenet/8.3.0' and its files, **libIngPKCS11.so** and **IngrianNAE.properties**, and should have read/write permissions. The directory and files within should not have 777, 776 or 773 permissions. They can have either the default permission (755) or 775 or 774.

6.	**Oracle Database - Enterprise Edition - Version 11.2.0.3 to 11.2.0.4 [Release 11.2] "HSM connection lost, closing wallet kzthsmterm: C_CloseSession threw PKCS11 error 48"**
	Note: *The information in this section applies to all platforms.*
	If TDE is used with HSM then brief interruptions in the connectivity to the HSM can cause errors on the database.
	Symptoms: Some symptoms observed in the `alert.log` are:
	`kzthsmcc4: HSM heartbeat died. Most likely connection has been lost. PKCS11 function C_Encrypt returned PKCS11 error code: 48`
	`HSM connection lost, closing wallet kzthsmterm: C_CloseSession threw PKCS11 error 48`
	`Sun Mar 02 17:16:54 2014`
	`kzthsmcc4: HSM heartbeat died. Most likely connection has been lost. PKCS11 function C_Encrypt returned PKCS11 error code: 48`
	`HSM connection lost, closing wallet kzthsmterm: C_CloseSession threw PKCS11 error 48`
	Changes: TDE is configured with HSM.
	Cause: The existing model is that the Oracle server initiates a heartbeat call to the HSM every 3 seconds from the gen0 process, if that fails the wallet will be closed, depending on the internal state of involved processes this can lead to various errors.
	Solution: To make the database more resilient to brief interruptions in the HSM connectivity a fix has been made for 11gR2.
	Patch 18948524: instance crash when HSM loses connectivity.
	Merge patches with this fix:
	Patch 19364977: MERGE REQUEST ON TOP OF DATABASE PSU 11.2.0.3.11 FOR BUGS 12874937 12951619
	Patch 19182734: MERGE REQUEST ON TOP OF DATABASE PSU 11.2.0.3.8 FOR BUGS 18511779 16360112
	Patch 19594366: MERGE REQUEST ON TOP OF DATABASE PSU 11.2.0.3.7 FOR BUGS 19148155 18948524
	(This list may not be exhaustive)
	Once any of these patches have been installed, that include the fix to bug 18948524, set the following events to control how the server will respond to the HSM failure, for example:
	event="28420 trace name context forever, level 10"
	event="28421 trace name context forever, level 3"
	To set this persistently, use the statement:

	ALTER SYSTEM SET EVENT='28420 trace name context forever, level 10:28421 trace name context forever, level 3' COMMENT='HSM heartbeat timeout and reconnect attempt' SCOPE=SPFILE; Event 28420 is used to determine how many HSM heartbeats can fail before the wallet is closed. The HSM heartbeat fires every 3 seconds which means that a very short network outage can lead to wallet closure. So, for example, if event 28420 is set to level 5 then the RDBMS will allow 5 heartbeats to fail before closing the wallet, which would allow a 15 second loss of contact with the HSM before closing the wallet. This event should NOT be used to disable the heartbeat functionality, and we strongly advise that this value is set no higher than 20 (i.e. one minute loss of HSM connectivity before wallet closure). Event 28421 will cause the HSM heartbeat to attempt to reconnect with the HSM once the wallet has been closed and, if successful, reopen the wallet. So, for example, if event 28421 is set to level 3 then every 3rd heartbeat will attempt to re-establish connection with the HSM. Event 28421 is only useful if HSM has been configured using the auto-open feature, this means a local `cwallet.sso` maintains the HSM credentials in the secret store entry `ORACLE.TDE.HSM.AUTOLOGIN`. The example uses the recommended values for the events. If the HSM is not available for prolonged periods eventually this may result in errors that cause the database to shutdown, this patch is only meant to cope with short outages. Also this patch is primarily meant to avoid a hard instance crash, if the HSM is not available then due to the current state of sessions doing any TDE related operation when the HSM connectivity is lost, those individual sessions may still get wallet related errors.

Oracle TDE Platform Matrix

The table below lists issues occurring with different Oracle versions on different platforms and their workarounds, wherever available.

#	Platform	Oracle Version	Known issue(s)	Workaround (if any)
1.	Windows 2008/2012 R2 64-bit	11.2.0.3 12.1.0.2	Wallet gets automatically closed after exiting the session.	This is an Oracle issue. No patch exists to resolve this issue.
			Database instance crashes on performing the following steps: Create an encrypted tablespace by executing `create tablespace`. Exit from the session. Open the wallet. Create another encrypted tablespace by executing `create tablespace`.	This is an Oracle issue. No patch exists to resolve this issue.

			Auto-login with HSM does not work.	This is an Oracle issue. No patch exists to resolve this issue.
2.	Windows 2003/2008 32-bit, 64-bit	11.2.0.2	Wallet gets automatically closed after some SQL operations.	Apply Oracle mandatory patch 11.2.0.2 bundle 15 (tracking patch numbers 13413154 (32bit) and 13413155 (64bit)). After applying the patch, use the shutdown immediate command to restart the database session.
		11.2.0.2 (with patch bundle 15)	Wallet gets automatically closed after exiting the session.	This is an Oracle issue. No patch exists to resolve this issue.
			Database instance crashes on performing the following steps: Create an encrypted tablespace by executing `create tablespace`. Exit from the session. Open the wallet. Create another encrypted tablespace by executing `create tablespace`	This is an Oracle issue. No patch exists to resolve this issue.
			Auto-login with HSM does not work.	This is an Oracle issue. No patch exists to resolve this issue.
3.	RHEL/AIX/ Solaris	11.2.0.2	Wallet gets automatically closed after some SQL operations.	Apply Oracle mandatory patch 12626642. After applying the patch, use the shutdown immediate command to restart the database session.

		11.2.0.2 (with patch 12626642), 11.2.0.3	When Oracle's database version 11.2.0.2 or beyond is used, encryption requests appear in the logs on the SafeNet KeySecure every three seconds. These encryptions are a result of Oracle's new heartbeat functionality related to TDE, specifically, related to external HSMs. If any of the heartbeats fails, Oracle immediately closes the connection with the HSM. This means that if one of the SafeNet KeySecure servers goes down or loses connectivity, SafeNet KeySecure's built-in failover/load balancing feature does not even get a chance to be used before the connection is closed, and wallet gets closed.	Enable Oracle Auto Login Wallet feature.
	SLES-12	12.1.0.2	Installation of oracle 12c release 12.1.0.2 on SLES-12 throws an error "[INS-13001] Environment does not meet minimum requirements".	This can either be ignored or suppressed by using below command for installation of oracle database: `./runInstaller -ignoreSysprereqs -ignorePrereq`