Oracle Transparent Data Encryption

nShield® HSM Integration Guide

11 Apr 2023

Contents

1. Introduction . . . 3
    1.1. Using this guide . . . 3
    1.2. Product configuration . . . 4
    1.3. Conventions used in this document . . . 5
    1.4. Overview . . . 8
2. Procedures . . . 10
    2.1. Preparatory requirements . . . 10
    2.2. Basic set up . . . 12
    2.3. Configuring Oracle database software to use the Entrust HSM . . . 14
    2.4. Opening and closing a keystore or HSM . . . 15
    2.5. Active credentials . . . 16
    2.6. Migrating from software wallet to HSM (non-multitenant) . . . 17
    2.7. Migrating from software keystore to HSM (multitenant) . . . 18
    2.8. Create master keys directly in an HSM for non-multitenant database . . . 19
    2.9. Create master keys directly in an HSM for multitenant database . . . 20
    2.10. Rekeying or key rotation . . . 22
3. Troubleshooting . . . 25
    3.1. An SQL command is run, and there is no output, or an unexpected output or error occurs . . . 25
    3.2. After a change to a configuration file, no resultant change in the database behavior is observed . . . 25
    3.3. ORA-28367: wallet does not exist . . . 25
    3.4. ORA-28367: cannot find PKCS11 library . . . 25
    3.5. ORA-28353: failed to open wallet . . . 26
    3.6. ORA-28407: Hardware Security Module failed with PKCS#11 error CKR_FUNCTION_FAILED (%d) . . . 26
    3.7. Encryption keys do not migrate correctly from a software keystore to an HSM (or vice-versa) . . . 26
    3.8. When you are using persistent OCS cards, the persistent authorization is lost . . . 26
    3.9. ORA-00600: internal error code . . . 27
    3.10. ORA-28374: Typed master key not found in wallet . . . 27
    3.11. ORA-12162: TNS: net service name is incorrectly specified . . . 27
4. Appendix . . . 28
    4.1. Security Worlds, key protection, and failure recovery . . . 28
    4.2. About the HSM credential . . . 30
    4.3. Change token with associated passphrase but keep same protection method . . . 34
    4.4. Latency issues . . . 36
    4.5. How Oracle works with the Entrust HSM . . . 40

1. Introduction

This guide describes how to integrate and use Entrust Security World software and Entrust Security nShield Hardware Security Modules (HSMs) with an Oracle database. The Oracle feature Transparent Data Encryption (TDE) provides data-at-rest encryption for sensitive information held by the Oracle database, while at the same time allowing authorized clients to use the database.

Oracle database software, and Entrust Security World software with nShield HSMs, can be independently installed on the same host server. They can then be configured to interoperate through a single library interface. It is possible to support multiple database instances on the same host server, while each database instance is restricted to access only its own encryption keys. Oracle cluster technology is also supported.

Integrated Oracle and Entrust technology has been tested to support Oracle TDE for tablespace encryption, or column encryption, or concurrently for both. Entrust nShield HSMs are certified to FIPS 140-2 (level 3) to deliver a high grade of security assurance. Functionality includes protection of sensitive enCONNECT TESTER@DB cryption keys and support for offload of encryption and key management operations.

This guides shows support for non-multitenant and multitenant databases. For Oracle version 21C, only multitenant Oracle database types are supported. Oracle does not support the creation of non-multitenant database types on version 21C. For more information on the multitenant support only by Oracle, see the Oracle multitenant documentation.

If using Oracle 18c or later, the `sqlnet.ora` file is officially deprecated and you should use the `WALLET_ROOT` and `TDE_CONFIGURATION` parameters.

1.1. Using this guide

This *Integration Guide* covers UNIX/Linux based systems. It provides:

- An overview of how the Oracle database software and Entrust Security World software with HSM work together to enhance security.
- Configuration and installation instructions.
- Depending on your current Oracle setup, how to:
  - Migrate encryption from an existing Oracle wallet or keystore to HSM protection.
  - Begin using HSM protection immediately if no Oracle software wallet or keystore already exists.

- Examples and advice on how the product may be used.
- Troubleshooting advice.

It is assumed the reader has a good knowledge of Oracle database technology.

Assuming you already have your Oracle database installed, after installing and configuring the Entrust Security World software with the HSM, there is no other software required. However, some minor configuration changes will be needed.

This guide cannot anticipate all configuration requirements a customer may have. Examples shown in this guide are not exhaustive, and may not necessarily show the simplest or most efficient methods of achieving the required results. The examples should be used to guide integration of the Entrust HSM with an Oracle database, and should be adapted to your own circumstances.

Entrust accepts no responsibility for loss of data, or services, incurred by use of examples, or any errors in this guide. For your own reassurance, it is recommended you thoroughly check your own solutions in safe test conditions before committing them to a production environment. If you require additional help in setting up your system, contact Entrust Support.

Entrust accepts no responsibility for information in this guide that is made obsolete by changes or upgrades to the Oracle product.

This guide assumes that you have read the Security World and HSM documentation, and are familiar with the documentation and setup processes for Oracle database TDE.

1.2. Product configuration

Entrust has successfully tested nShield HSM integration with the in the following configurations:

OS Version	Kernel	Oracle Version
Red Hat Enterprise Linux 8.7 (Ootpa)	Linux 4.18.0-240.el8.x86_64	Oracle Database 21c Enterprise Edition - 21.3.0.0.0
Red Hat Enterprise Linux 8.7 (Ootpa)	Linux 4.18.0-425.10.1.el8_7.x86_64	Oracle Database 19c Enterprise Edition - 19.17.0.0.0

1.2.1. Supported nShield hardware and software versions

Entrust has successfully tested with the following nShield hardware and software versions:

1.2.1.1. Oracle 21C - 21.3.0.0.0

HSM	Security World Software	Firmware	Image	OCS	Softcard	Module	FIPS Level 3
Connect XC	12.80.4	12.50.11 (FIPS Certified)	12.80.4	✓	✓	✓	
Connect XC	12.80.4	12.72.1 (FIPS Certified)	12.80.5	✓	✓	✓	✓
nShield 5	13.2.2	13.2.2 (FIPS Pending)	13.2.2	✓	✓	✓	

1.2.1.2. Oracle 19C - 19.17.0.0.0

HSM	Security World Software	Firmware	Image	OCS	Softcard	Module	FIPS Level 3
Connect XC	12.80.4	12.50.11 (FIPS Certified)	12.80.4	✓	✓	✓	
Connect XC	12.80.4	12.72.1 (FIPS Certified)	12.80.5	✓	✓	✓	✓
nShield 5	13.2.2	13.2.2 (FIPS Pending)	13.2.2	✓	✓	✓	

1.3. Conventions used in this document

1.3.1. Multitenant and non-multitenant

Descriptions in this *Integration Guide* may cover non-multitenant databases and multitenant databases. Keep in mind that creation of non-multitenant databases are not supported anymore from Oracle 21C. This guide will use the terms appropriate to the database type under discussion, as outlined:

- Non-multitenant databases are on Oracle version 11g or earlier. Multitenant databases start from Oracle version 12c.
- Non-multitenant database software can only create and use non-multitenant databases. If non-multitenant databases are the subject matter, use the non-

multitenant and SQL terminology as shown below.

- Database software supporting multitenant databases may also optionally support non-multitenant databases (pre-21c). In this case, if a non-multitenant mode is the subject matter, then use the non-multitenant terminology and SQL shown below. If a multitenant mode is the subject matter, then use the multitenant terminology and SQL.
- Non-Multitenant (non-container)
  1. Terminology for Oracle software based encryption key repository.

     ALTER SYSTEM SET ENCRYPTION ...
- Multitenant (container)
  1. Terminology for Oracle software based encryption key repository

     Software keystore
  2. SQL preamble for encryption related commands

     ADMINISTER KEY MANAGEMENT, etc

Where such terminology applies equally to a software wallet or software keystore, the default terminology software keystore is used to cover both descriptive instances.

1.3.2. Database connections

You must be a user with correct permissions to access a database, and also have the correct privileges to perform the required operations when connected to that database. Your system administrator should be able to create users and grant suitable permissions and privileges according to your organization's security policies. Example 2

- `<database-user>` is the user identity making the connection.
- `<database-identifier>` is the database to make the connection to.

For the purpose of examples in this guide, the following database users and database identifiers should be sufficient.

- `<database-user>`. This guide will use one following users for connecting to databases:
  - `sysdba`, Oracle's standard sysdba user.
  - `system`, Oracle's standard system user.
  - Non-Mutitenant:
    - `TESTER`, as a local user.
  - Multitenant:
    - `C##TESTER`, as a common user for container (CDB) and the PDBs it contains.

        - CDB<n>PDB<k>TESTER, as a local user for a PDB<k> within container CDB<n>.

          Where <n> and <k> are distinguishing digits.

- <database-identifier>. This guide will use one following database identifies during a connection:
    - Non-Multitenant databases:
        - DB, in practice usually the ORACLE_SID of the database. For example:

```
CONNECT sysdba@DB
CONNECT TESTER@DB
```

    - Multitenant databases:

        - CDB<n> indicates a container database where <n> is a distinguishing digit.
        - PDB<k> indicates a pluggable database where <k> is a distinguishing digit.

      Multitenant database identifiers will be:

        - CDB<n>, to connect to the $CDB<n>$ROOT for a particular container CDB<n>.
        - CDB<n>PDB<k>, to connect to PDB<k> within CDB<n>.

      For example:

```
CONNECT sysdba@CDB1
CONNECT C##TESTER@CDB1
CONNECT C##TESTER@CDB1PDB2
CONNECT CDB1PDB1TESTER@CDB1PDB1
```

When you are using a multitenant database, the connection implies that you must alter a session if you are not already connected to the required container. For example:

- Example 1:

  CONNECT C##TESTER@CDB<n>

  This implies that, if you are not already connected to CDB<n>, then alter the session:

  ALTER SESSION SET CONTAINER = CDB<n>$ROOT;

- Example 2:

  CONNECT CDB<n>PDB<k>TESTER@CDB<n>PDB<k>

  This implies that, if you are not already connected to CDB<n>PDB<k>, then alter the session:

  ALTER SESSION SET CONTAINER = CDB<n>PDB<k>;

Examples of `sqlplus` connection syntax for different users:

- sqlplus / as sysdba
- sqlplus / as sysdba@CDB1ROOT
- sqlplus CDB1PDB1TESTER/Tester@//localhost:1521/CDB1PDB1.interop.com

1.3.3. Key migration and legacy keys

Encryption master keys may be migrated from an existing Oracle keystore to an Entrust HSM, or vice versa. In this case, and as used in this document, the term 'key migration' means that the responsibility for holding the master keys is being migrated. The encryption keys themselves are not copied (or imported) between a software keystore and HSM Security World. Fresh master key(s) are created within the software keystore or HSM that is to become the new key protector as a result of the migration. Subsidiary keys that are being protected are re-encrypted using the fresh master key(s). Thereafter, any new master keys are created in the current key protector you have migrated to.

During rekey, the previous master keys, or legacy keys, remain in the software keystore or HSM where they were created. After you have performed a key migration, you can retain access to the legacy keys in the software keystore or HSM you have migrated away from by making its passphrase the same as the current key protector's. This allows both to be open at the same time allowing access to encryption keys they both contain. If you do not do this, you will only be able to access keys in the current key protector. If you are using both a software keystore and HSM at the same time, whichever is the current key protector is called the primary.

1.4. Overview

Transparent Data Encryption (TDE) is used to encrypt an entire database in a way that does not require changes to existing queries and applications. A database encrypted with TDE is automatically decrypted when the database loads it into memory from disk storage, which means that a client can query the database within the server environment without having to perform any decryption operations. The database is encrypted again when saved to disk storage. When using TDE, data is not protected by encryption whilst in memory. The encryption keys that are used to encrypt the database are typically held as part of the database, but these keys are themselves encrypted using a master encryption key in order to protect them. Using an Entrust HSM allows the master encryption keys to be kept physically separate from the database it is protecting, and also provides a hardware protected boundary from which encryption keys can never leave in plaintext. Additionally, the encryption keys are held in a Security World folder which is also encrypted and is useless to anyone who does not possess the authorized means to access them. The Security World folder permits easy back up or transfer to

other legitimate clients that may use the authorized mechanisms to access the encryption keys.

Other benefits of using the nShield HSM include:

- Ability to store keys from all across an enterprise in one place for easy management.
- Key Retention (rotate keys while keeping the old ones).
- FIPS and Common Criteria compliance.

2. Procedures

2.1. Preparatory requirements

Before installing the software, Entrust recommends that you familiarize yourself with:

- The Oracle database TDE documentation and setup process.
- The Entrust documentation.

Entrust also recommends you have an agreed organizational Certificate Practices Statement and a Security Policy/Procedure in place covering administration of the HSM. In particular, these documents should include the following aspects of HSM administration:

- Whether the Security World must comply with FIPS 140-2 Level 3 or Common Criteria restrictions.
    - If you want to use a FIPS 140-2 Level 3 Security World, then you must create an OCS card set for FIPS authorization. This is true even if you want to use module or Softcard protection.
    - If you are running multiple database instances on the same host, the same FIPS authorizing OCS cards can be used for all database instances.
    - If you want to use OCS protection, the same OCS card set used for key protection can also be used for FIPS authorization.
- The number and quorum of Administrator Cards in the Administrator Card Set (ACS), and a policy for managing these cards.
- Which of the following Entrust encryption key protection methods you want to use:
    - Module protection
    - Softcard protection
    - Operator Card Set (OCS) protection.

      If OCS cards are to be used, you need to decide the number of Operator Cards in the OCS card set. K/N functionality is not currently supported. This means that you must create 1/N OCS card sets. The number of OCS cards in a card set must at least match the number of HSMs that will be in your configuration, and with more to spare in case of a card loss or failure.
- Entrust recommends that you create a policy for managing SQL scripts that allow use of credentials for the Oracle database. These SQL scripts should only be available to authorized users.
- Entrust recommends that you create a policy for managing the passphrases for your:

    - ACS
    - Module protection
    - Softcard protection
    - OCS protection

  For information on passphrases, see About the HSM credential.

- Entrust recommends that you create a policy for managing the physical security of your smartcards as used for ACS and OCS, and their deployment to authorized users.

As part of your preparation, Entrust recommends that you read Security Worlds key protection and failure recovery.

This guide assumes that Oracle database software, and (at least) one Oracle database, is already installed on your system. With Oracle database software already installed, ensure that any required patches have been added.

To integrate an Oracle database with an Entrust HSM, the following steps are required:

1. Environment configuration.
2. Install the Entrust HSM and Security World software.
3. Configure Oracle database software to use the Entrust HSM.

Details of your installation and configuration will depend on:

- Whether you are using a non-multitenant or multitenant database.
- Whether you want to migrate encryption keys from an existing Oracle software keystore to an Entrust HSM, or start directly with an Entrust HSM.

The default host server user is `oracle` unless stated otherwise.

For more information on how to configure your Entrust environment, see the *User Guide* for your HSM.

For more information on how to configure your Oracle environment, see the Oracle documentation.

For more detail or suggestions on how you may set up your system, see the following Appendixes:

- Security Worlds key protection and failure recovery.
- About the HSM credential.
- Latency issues.

2.2. Basic set up

1. Install the Entrust Security World software on each client in accordance with its accompanying documentation. If you are using Entrust Connects with a separate RFS, the Entrust Security World software must also be installed on the RFS.
2. Create or edit the `cknfastrc` file located in the `NFAST_HOME` directory for each client, and depending on how you want to protect the master encryption key(s), set the following PKCS#11 environment variables:

    - Including OCS or Soft card key protection, and HSM load sharing:

    ```
    CKNFAST_LOADSHARING=1
    ```

    - Including module key protection:

    ```
    CKNFAST_FAKE_ACCELERATOR_LOGIN=1
    ```

    For more information, study the PKCS#11 library environment variables in the *User Guide* for your HSM.

3. If you are using Entrust Connect(s), configure these to operate with your selected RFS and client(s) as described in your HSM documentation. Typically the client(s) will be the host server that your Oracle database is running on.
4. Configure the Oracle PKCS#11 library folder to use the Entrust PKCS#11 API.

    After creating the Oracle database, you will have to:

    a. Create the following directory path for the Entrust API library as the `oracle` user:

    Make ownership and permissions on the directory as: `owner=oracle; group=oinstall; permissions=775`.

    ```
    mkdir -p $ORACLE_BASE/extapi/64/hsm/nCipher/12.80.4
    chown oracle $ORACLE_BASE/extapi/64/hsm/nCipher/12.80.4
    chgrp oinstall $ORACLE_BASE/extapi/64/hsm/nCipher/12.80.4
    chmod 775 $ORACLE_BASE/extapi/64/hsm/nCipher/12.80.4
    ```

    b. copy/link the PKCS#11 library into the directory as the `oracle` user.

    ```
    cp /opt/nfast/toolkits/pkcs11/libcknfast.so $ORACLE_BASE/extapi/64/hsm/nCipher/12.80.4
    ```

The Entrust PKCS#11 API library is the only means by which the Oracle database system can communicate with the Entrust system. If this interface is not set up correctly, you will not be able to get these two systems to operate together.

5. Add the `oracle` user to the `nfast` group.

```
sudo usermod -a -G nfast oracle
```

2.2.1. Security World creation

1. Create or load the Security World using a client, or nShield Connect (if being used). If you are using RA for the ACS cards, you must do so through a registered client. Ensure the Security World data is copied to the `NFAST_KMDATA/local` folder for all clients and the RFS, and is loaded onto each nShield Connect used in the configuration.
2. Check the Security World on your various components as follows:
   - Client: Use the Entrust `nfkminfo` utility to check the Security World and configuration on each client. In each case, the Security World must be shown as **Initialized** and **Usable**.
   - RFS: Use the Entrust 'nfkminfo' utility to check the Security World and configuration. The Security World must be shown as **Initialized**.
   - nShield Connect:
     - Front panel: **MENU > Security World mgmt. > Display World Info**.

       The Security World must be shown as **Initialized** and **Usable**.

     - If you are using Security World software v12, on the client run the Entrust `nethsmadmin` utility:

       ```
       >>nethsmadmin -c -m<n>
       ```

       Where `<n>` is the module number. The Security World must be shown as **Initialized** and **Usable**. For further details, see the *User Guide* for your HSM.

2.2.2. Prepare protection method

1. If your Security World does not already contain the required protection method, then proceed as follows:
   - If you want to use module protection, no action is required at this point. Action is required later in the integration.
   - If you want to use Softcard protection, create the required number of Softcard(s), each with its own passphrase.
   - If you want to use a 1/N OCS card set protection, create the required number of card set(s) now, using exact same passphrase for each card within the same card

set. See About the HSM credential.

2. If you are using module or Softcard protection in a FIPS 140-2 Level 3 environment, then you also need an OCS card set (1/N) to provide FIPS authorization. If a suitable OCS card set is not already available in the Security World, then create an OCS card set for this purpose.

2.3. Configuring Oracle database software to use the Entrust HSM

Before proceeding, it is assumed that:

- You have followed the set up and configuration instructions in this guide. That is:
  - The Oracle database software is installed with at least one database instance.
  - The Entrust Security World software and HSM are installed and configured.
  - Your protection method has been prepared.
- The target container database (CDB) is open, and all PDBs are open.

You can use the following instructions to configure your Oracle database software to function using the Entrust HSM and Security World software, in one of the following scenarios:

- Migration from keystore to HSM: One or more database instances are already using TDE encryption, each instance with its own software keystore, and you want to continue using TDE encryption after migrating the TDE master keys from at least one keystore to the Entrust HSM.
- Create keys directly in HSM: One or more database instances are not using TDE encryption, and you want to start using TDE encryption for at least one database, using the Entrust HSM.

Before attempting key migration, see Key migration and legacy keys. Oracle 11.1g or earlier versions might not support migration of some key types from a software wallet to an HSM. See the documentation for your Oracle version before attempting key migration.

The SQL commands that will be used later in this document might:

- Require more than one user with suitable database privileges to make the specific database connections, and run the SQL commands in the sequences as shown. Respect the connections shown in order to satisfactorily run SQL on your target. See Database connections. Your system administrator should have sufficient knowledge to create users and associated privileges according to your organization's security policies.
- Need to be run as a certain user. If you are instructed in this guide to make a connection as a particular user, continue with that connection until instructed

otherwise.

- Use <credential> to denote your chosen protection method. When a protection method has been invoked, you must continue with the same protection method unless you decide to alter it as described in About the HSM credential.

Oracle documentation uses the <credential-name>|<credential-passphrase> order. However, tests showed that the ordering <credential-passphrase>|<credential-name> works. In SQL, the credential used to open a keystore must match the credential used to create an encryption key.

Whenever you have completed migrating or creating encryption keys in an HSM, it is recommended to back up your Security World data, see the *User Guide* for your HSM.

Make sure you use instructions appropriate to whether you are using:

- A non-multitenant database and software wallet.
- A multitenant database and software keystore.

2.4. Opening and closing a keystore or HSM

Oracle has a control system that gates access to a software keystore or HSM:

- If a keystore or HSM is open, then you can access its contents.
- If a keystore or HSM is closed, then you cannot access its contents.

You can open or close a software keystore or HSM with the following SQL statements.

2.4.1. Non-multitenant

This section assumes database is open.

- To open/close wallet:

  CONNECT TESTER@DB or CONNECT sysdba@DB

```
--To open wallet
ALTER SYSTEM SET [ENCRYPTION] WALLET OPEN IDENTIFIED BY "<credential>";

--To close wallet, pre-11.2.0.1.0
ALTER SYSTEM SET [ENCRYPTION] WALLET CLOSE;

--To close wallet, 11.2.0.1.0 onward
ALTER SYSTEM SET [ENCRYPTION] WALLET CLOSE IDENTIFIED BY "<credential>";
```

Where the [ENCRYPTION] clause is optional.

2.4.2. Multitenant

This section assumes the respective CDB and PDB databases are open:

- To open/close keystore for the container (CDB) only.

  CONNECT C##TESTER@CDB<n>

  ```
  ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN IDENTIFIED BY "<credential>";
  ADMINISTER KEY MANAGEMENT SET KEYSTORE CLOSE IDENTIFIED BY "<credential>";
  ```

- To open/close keystore for the container (CDB) and all PDBs it holds.

  CONNECT C##TESTER@CDB<n>

  ```
  ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN IDENTIFIED BY "<credential>" CONTAINER=ALL;
  ADMINISTER KEY MANAGEMENT SET KEYSTORE CLOSE IDENTIFIED BY "<credential>" CONTAINER=ALL;
  ```

  If you want to close all keystores, use the following SQL:

  ```
  ADMINISTER KEY MANAGEMENT SET KEYSTORE CLOSE CONTAINER=ALL;
  ```

- To open/close keystore for a single PDB, you must use same credential as used by the containing CDB.

  CONNECT PDB<k>TESTER@CDB<n>PDB<k>

  ```
  ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN IDENTIFIED BY "<credential>";
  ADMINISTER KEY MANAGEMENT SET KEYSTORE CLOSE IDENTIFIED BY "<credential>";
  ```

2.4.2.1. Issues closing keystores

During migration from Software Wallet to HSM Keystore, you may experience issues closing the keystore. To resolve this, disable the auto-login keystore to close all keystores. See How To Disable Auto-Login Keystore for full details.

```
sudo -u oracle mv <path-to-keystorefolder>/<keystore-folder>/tde/cwallet.sso <path-to-keystorefolder>/<keystore-folder>/tde/cwallet.sso.backup
```

2.5. Active credentials

The first time you open a keystore or HSM using a credential for a particular database

instance, it activates the credential you are referencing. You should then be able to create master encryption keys, or use (any) existing master encryption keys, that are protected by that credential. You cannot have more than one active credential at the same time for the same instance. You must close the keystore or HSM to deactivate the credential.

You can simultaneously use different credentials for different database instances on the same host server. For a container database only its CDB is a real instance. All PDBs within the same CDB must use the same active credential.

See About the HSM credential if you want to change a credential.

2.6. Migrating from software wallet to HSM (non-multitenant)

The following procedure applies when the target database is non-multitenant, and you are already using a software wallet with TDE encryption. If your target database is multitenant, see Migrating from software keystore to HSM (multitenant).

Entrust strongly recommends you back up your software wallet as an independent operation before attempting migration to the HSM. Keep the backup folder in a safe place separated from the associated database files. Only users with authorization should be able to access the backup folder.

Repeat the following procedure for each database software wallet from which you want to migrate. Each independent database instance can use its own Entrust key protection method or credential if required.

Once an Entrust key protection method has been activated for a particular database instance, then you must continue to use that same credential for any further keys you want to protect for that instance.

See About the HSM credential if you want to change a credential.

Use the `WALLET_ROOT` and `TDE_CONFIGURATION` parameters. It is assumed the `WALLET_ROOT` parameter has already been set for Oracle keystore use.

1. Prepare for key migration by running the following SQL script:

   CONNECT sysdba@DB

   ```
   ALTER SYSTEM SET TDE_CONFIGURATION = "KEYSTORE_CONFIGURATION=HSM|FILE" SCOPE=BOTH SID='*';
   ```

2. Migrate from the keystore to HSM:

   CONNECT sysdba@DB

```
ADMINISTER KEY MANAGEMENT SET ENCRYPTION KEY IDENTIFIED BY <credential> MIGRATE USING <keystore-passphrase> WITH
BACKUP;
```

Use the Entrust `rocs` utility to check that your encryption keys have been stored under the expected protection method before proceeding.

2.7. Migrating from software keystore to HSM (multitenant)

The following procedure applies when the target database is multitenant, and you are already using a software wallet with TDE encryption. If your target database is non-multitenant, see Migrating from software wallet to HSM (non-multitenant).

Repeat the following procedure for each software keystore from which you want to migrate. Each container database (CDB) can use its own Entrust key protection method (credential) if required. However, once a Entrust key protection method has been activated for a particular database instance (CDB), then you must continue to use that same credential for any further keys you want to protect for that instance.

See About the HSM credential if you want to change a credential.

Use the `WALLET_ROOT` and `TDE_CONFIGURATION` parameters.

1. Back up your software keystore before attempting key migration to the HSM:

   CONNECT sysdba@CDB<n>

   ```
   ADMINISTER KEY MANAGEMENT BACKUP KEYSTORE USING '<PreMigrationBackupString>' IDENTIFIED BY "<keystorepassphrase>";
   ```

2. Prepare for key migration by running the following SQL script:

   CONNECT sysdba@CDB1ROOT

   ```
   ALTER SYSTEM SET TDE_CONFIGURATION = "KEYSTORE_CONFIGURATION=HSM|FILE" SCOPE=BOTH SID='*';
   ```

3. Create an auto-login keystore where `<credential>` is the HSM credential you want to use:

   CONNECT sysdba@CDB1ROOT

```
ALTER PLUGGABLE DATABASE ALL OPEN;
ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN IDENTIFIED BY <keystore-passphrase> CONTAINER = ALL;
ADMINISTER KEY MANAGEMENT ADD SECRET "<credential>" FOR CLIENT 'HSM_PASSWORD' IDENTIFIED BY <keystore-passphrase>
WITH BACKUP;
ADMINISTER KEY MANAGEMENT CREATE AUTO_LOGIN KEYSTORE FROM KEYSTORE <path-to-keystorefolder>/<keystore-folder>/tde'
IDENTIFIED BY KeystorePassword1;
```

4. Migrate from the keystore to HSM:

   CONNECT sysdba@CDB1ROOT

```
ADMINISTER KEY MANAGEMENT SET ENCRYPTION KEY IDENTIFIED BY "<credential>" MIGRATE USING <keystore-passphrase> WITH
BACKUP;
```

Use the Entrust `rocs` utility to check that your encryption keys have been stored under the expected protection method before proceeding.

2.8. Create master keys directly in an HSM for non-multitenant database

The following procedure applies when the target database is non-multitenant, and there is no pre-existing software wallet. If your target database is multitenant, see Create master keys directly in an HSM for multitenant database.

Repeat the following procedure for each database in which you want to create keys. Each database can use its own Entrust key protection method (credential) if required. However, once an Entrust key protection method has been activated for a particular database instance, then you must continue to use that same credential for any further keys you want to protect for that instance.

See About the HSM credential if you want to change a credential.

2.8.1. Use the WALLET_ROOT and TDE_CONFIGURATION parameters

1. Set up the `WALLET_ROOT` and `TDE_CONFIGURATION` parameters as follows. You must set up the `WALLET_ROOT` parameter even if you do not use a keystore.

   CONNECT sysdba@DB

```
ALTER SYSTEM SET WALLET_ROOT = '<path-to-keystore>' scope=SPFILE;
```

2. Bounce the database after setting up the `WALLET_ROOT` parameter.

   CONNECT sysdba@DB

```
ALTER SYSTEM SET TDE_CONFIGURATION = "KEYSTORE_CONFIGURATION=HSM" SCOPE=BOTH SID='*';
```

3. Bounce the database after setting up the `TDE_CONFIGURATION` parameter.

2.8.2. Create the encryption keys

1. Select the protection method (credential) that you require below, and run the SQL.

   CONNECT TESTER@DB or CONNECT sysdba@DB

```
ALTER SYSTEM SET ENCRYPTION KEY IDENTIFIED BY "<credential>";
```

Use the Entrust `rocs` utility to check that your encryption keys have been stored under the expected protection method before proceeding.

After you created the master encryption keys in the HSM as above, proceed to encrypt your database by using tablespace encryption, column encryption, or both, as usual.

2.9. Create master keys directly in an HSM for multitenant database

The following procedure applies when the target database is multitenant, and there is no preexisting software keystore. If your target database is non-multitenant, see Create master keys directly in an HSM for non-multitenant database.

Repeat the following procedure for each database in which you want to create keys. Each database instance can use its own Entrust key protection method (credential) if required. However, once an Entrust key protection method has been activated for a particular database instance (CDB), then you must continue to use that same credential for any further keys you want to protect for that instance.

See About the HSM credential if you want to change a credential.

You must create the container (CDB) master key first. After the CDB master key has been created you have a choice of creating master keys for all the PDBs it contains in one operation, or else for each PDB individually.

The PDB(s) must use the same protection credential as the CDB.

2.9.1. Use the WALLET_ROOT and TDE_CONFIGURATION parameters

1. Set up the `WALLET_ROOT` and `TDE_CONFIGURATION` parameters as follows. You must set up

the WALLET_ROOT parameter even if you do not use a keystore.

CONNECT sysdba@CDB1ROOT

```
ALTER SYSTEM SET WALLET_ROOT = '<path-to-keystore>' scope=SPFILE;
```

2. Bounce the database after setting up the WALLET_ROOT parameter.
3. Run the following command:

```
ALTER SYSTEM SET TDE_CONFIGURATION = "KEYSTORE_CONFIGURATION=HSM" SCOPE=BOTH SID='*';
```

4. Bounce the database after setting up the TDE_CONFIGURATION parameter.

2.9.2. Create the CDB and then all PDB master keys in one operation

1. Select the protection method you require below, and run the SQL:

   CONNECT C##TESTER@CDB<n>

```
ALTER PLUGGABLE DATABASE ALL OPEN;

--This will activate the credential
ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN IDENTIFIED BY "<credential>" CONTAINER=ALL;
```

2. Activate master keys for the CDB and all the PDBs in one operation:

```
ADMINISTER KEY MANAGEMENT SET KEY IDENTIFIED BY "<credential>" WITH BACKUP CONTAINER=ALL;
```

Use the Entrust rocs utility to check that your encryption keys have been stored under the expected protection method before proceeding.

Encrypt your database using tablespace encryption, column encryption, or both.

2.9.3. Create the CDB master key and a single PDB master key

1. Create the CDB master key:

   CONNECT C##TESTER@CDB<n>

   a. Select the protection method you require below, and run the SQL:

```
--This will activate the credential if it isn't already
ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN IDENTIFIED BY "<credential>";
ADMINISTER KEY MANAGEMENT SET KEY IDENTIFIED BY "<credential>" WITH BACKUP;
```

b. Once you have created the CDB master key, you can repeat the following commands for creating a single PDB master key, for any PDB you select.

2. Create a single PDB master key:

   CONNECT PDB<k>TESTER@CDB<n>PDB<k>

   You must use the same protection method (credential) as the containing CDB. Run the SQL.

```
--If the PDB is already open, you don't need to do this.
ALTER PLUGGABLE DATABASE <CDB<n>PDB<k>> OPEN READ WRITE;

--If the keystore is already open, you don't need to do this.
ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN IDENTIFIED BY "<credential>";

--Make the master key for the PDB you should be currently connected to.
ADMINISTER KEY MANAGEMENT SET KEY IDENTIFIED BY "<credential>" WITH BACKUP;
```

Use the Entrust `rocs` utility to check that your encryption keys have been stored under the expected protection method before proceeding.

Encrypt your database using tablespace encryption, column encryption, or both.

2.10. Rekeying or key rotation

After you have established your HSM as the primary protector for your master encryption keys, for security reasons you may want to periodically replace the keys, or rekey. For your particular system, you can do this by following the instructions below.

The following subsections show how to perform a rekey in Oracle multitenant environments. After rekey, the new encryption keys should be immediately available and usable by the client that initiated the rekey.

2.10.1. Rekey when sharing keys between clients

If the encryption keys are being shared or distributed between clients, then either a common shared Security World folder, or local client copies of the Security World folder, will be used. In this case, you must factor in:

- Encryption key distribution and synchronization with the associated encrypted data in the Oracle database.
- Recognition of new encryption keys by the Entrust hardserver instance on each client.

For the new keys to be recognized by a client hardserver instance (that did not initiate the rekey), you must first make sure that the new keys are available in the Security World

folder it is using. If the new keys are available, then you can make the client hardserver instance recognize and use the new keys using either of the following options:

1. In the client `cknfastrc` file, set an environment variable:

```
CKNFAST_ASSUME_SINGLE_PROCESS=0
```

2. Reconnect all users/applications on the client that are using the database encryption facilities.

The above actions will cause the available keys to be scanned by the client's hardserver instance, and any new keys will then be recognized and made usable. See Latency issues to understand the full consequences of these options.

It is the job of your system administration to ensure that distribution and recognition of shared (new) encryption keys is performed smoothly. In the (unlikely) event that synchronization problems cannot be resolved with the system in continual operation, it may be necessary to temporarily halt encrypted database operations on all clients other than the one that initiated the rekey. After rekey has been performed, with correct keys available and recognized by all clients, then the system can be restored to normal operations.

Test your rekey arrangements in a safe environment before committing to a production environment. Transactions restricted to unencrypted data will not be affected by rekey operations.

Before rekeying, you should inspect the contents of your Security World `local` folder, and note the date/time that you perform a rekey. After rekeying, you should verify that new key files have been created in your Security World `local` folder by inspection, and check the date/time stamp of new key files in the folder match the date/time you performed the rekey.

2.10.2. Rekey for a non-multitenant database

The following instructions begin by assuming the HSM (wallet) is already open.

CONNECT TESTER@DB, or CONNECT sysdba@DB

```
--Assumes HSM is already open
ALTER SYSTEM SET ENCRYPTION KEY IDENTIFIED BY "<credential>";
```

2.10.3. Rekey for a multitenant database with CDB and all the PDBs in one operation

CONNECT TESTER@CDB<n>

The following instructions begin by assuming the required CDB has started, and required PDBs and HSM (keystore) to be already open.

```
ADMINISTER KEY MANAGEMENT SET KEY IDENTIFIED BY "<credential>" WITH BACKUP CONTAINER=ALL;
```

2.10.4. Rekey for a multitenant database with CDB only

The following instructions begin by assuming the required CDB has started and HSM (keystore) to be already open.

CONNECT TESTER@CDB<n>

```
ADMINISTER KEY MANAGEMENT SET KEY IDENTIFIED BY "<credential>" WITH BACKUP;
```

2.10.5. Rekey for a multitenant database with a single PDB only

The following instructions begin by assuming the required CDB has started, the required PDB and HSM (keystore) to be already open.

CONNECT PDB<k>TESTER@CDB<n>PDB<k>

```
--Make the master key for the PDB you should be currently connected to
ADMINISTER KEY MANAGEMENT SET KEY IDENTIFIED BY "<credential>" WITH BACKUP;
```

3. Troubleshooting

Oracle error messages may sometimes show error symptoms rather than the root cause. If you see an error you have not encountered before, search for further information online before attempting to resolve the error. If you remain unable to resolve the error, contact Oracle support.

If you edit an Oracle configuration file, use a simple text editor running on the host. Do not cut and paste the file contents from another file using a formatting editor, as it may insert hidden characters that are difficult to detect and which can stop the file from working. Entrust also suggests you avoid copying files onto a UNIX host via a Windows intermediary (this includes library files).

3.1. An SQL command is run, and there is no output, or an unexpected output or error occurs

1. Try reconnecting to the database.
2. If that does not work, try bouncing the database.

3.2. After a change to a configuration file, no resultant change in the database behavior is observed

1. Try reconnecting to the database.
2. If that does not work, try bouncing the database.

3.3. ORA-28367: wallet does not exist

1. Check that you have correctly installed and configured the Entrust PKCS#11 library.
2. Try reconnecting to the database.
3. Try bouncing the database.
4. Try restarting the Entrust hardserver.

3.4. ORA-28367: cannot find PKCS11 library

1. Ensure that you have correct permissions to use the `/opt/oracle/extapi/⋯` directory.
2. Check that you are using a library for the correct local architecture (32/64).
3. Check that you are using the appropriate Java version (32/64).
4. Refer to advice given above about editing Oracle files, or copying them.
5. Try reconnecting to the database.

6. Recopy the `libcknfast.so` library file to `/opt/oracle/extapi/`.
7. In the `ORACLE_BASE/extapi` directory, create a link named `libcknfast.so` to the actual `NFAST_HOME/toolkits/pkcs11/libcknfast.so` file.

3.5. ORA-28353: failed to open wallet

1. Check that you have set up your `cknfastrc` file with the correct contents.
2. Ensure that the HSM wallet pass phrase is correct.
3. Ensure that if OCS/Softcard key protection is used, the name and passphrase are correct and are separated by a **|** or a **:**.
4. If you have migrated from an Oracle wallet to an HSM wallet, you must update the passphrase.

3.6. ORA-28407: Hardware Security Module failed with PKCS#11 error CKR_FUNCTION_FAILED (%d)

1. This may be caused by Oracle defect 23528412. Contact Oracle support in order to obtain a patch for this defect.
2. Ensure that if a FIPS 140-2 Level 3 Security World is in use, an OCS card is inserted in the HSM slot.
3. Check that you are using the correct passphrase/credential to access the HSM.
4. If you are using an nShield Connect, use its front panel to check the Security World is loaded on to the HSM itself and is both **Initialized** and **Usable**.
5. Try restarting the Entrust hardserver.

3.7. Encryption keys do not migrate correctly from a software keystore to an HSM (or vice-versa)

1. This may be caused by Oracle defect 17409174. Contact Oracle support in order to obtain a patch for this defect.

3.8. When you are using persistent OCS cards, the persistent authorization is lost

1. This may be caused by Oracle defect 23528412. Contact Oracle support in order to obtain a patch for this defect.
2. Ensure that, as the required OS user, you can access both the Entrust and Oracle functionality. If necessary, adjust user group membership to permit this, but check your security policy first.

3.9. ORA-00600: internal error code

3.9.1. arguments: [kzthsmgmk: C_GenerateKey], [6], [],[], [], [], [], []

1. Ensure that you have added the `oracle` user to the `nfast` group. In some cases, you may have to re-login with the `oracle` user for this to take effect.
2. Ensure that if a FIPS 140-2 Level 3 Security World is in use, an OCS card is inserted in the HSM slot.

3.9.2. arguments: [ksqgel:null_parent], [], [],[], [], [], [], []

1. Sometimes occurs using encrypted tablespaces.
2. This may be caused by Oracle defect 21080143. Contact Oracle support in order to obtain a patch for this defect.

3.10. ORA-28374: Typed master key not found in wallet

1. Oracle software thinks there is a mismatch between encrypted object(s) and available master key(s). There is more than one possible cause for this and it is usually quite difficult to resolve. Contact Oracle support, or search for a solution online.
2. If all else fails, try and restore your system from backups.

3.11. ORA-12162: TNS: net service name is incorrectly specified

1. Check that you have correctly set the value for ORACLE_SID in your local environment.

4. Appendix

4.1. Security Worlds, key protection, and failure recovery

This section highlights some considerations when choosing Security World and key protection options for use with the Entrust Security World. It focuses on recovery of Security World authorization where a system has temporarily failed (for instance after a power outage) and is then returned to operation. This does not apply to other failure recovery functions. These considerations are applicable to Security Worlds, key protection and failure recovery for both standalone systems and database clusters. For a fuller explanation of Security Worlds and key protection, refer to the *User Guide* for your HSM.

In the event of a temporary failure of the Entrust Security World, there may be a consequent loss of:

- Credential authorization.
- Authorization if you are using a FIPS 140-2 Level 3 Security World.

A credential authorization can be granted using either a Softcard or an OCS card, with passphrase. In the case of an OCS, a card must be always available in a valid HSM card reader in order to grant reauthorization after a failure, and permit automatic recovery.

Where FIPS authorization is required, this can be granted either by using an OCS card specifically for this purpose, or through an OCS card that is also used for credential authorization. A card from the OCS must be always available in a valid HSM card reader in order to grant reauthorization after a failure, and permit automatic recovery.

If you are using OCS cards through a RA secure channel, then if the secure channel is lost it must be reestablished before recovery using the OCS cards can begin. There is no automatic mechanism to reestablish the secure channel, which would have to be re-established manually, or through a user-defined script. For this reason, Entrust recommends that RA is not used for systems requiring automatic recovery.

Oracle auto-login facilities need to be set up to implement automatic recovery in the event of a temporary failure.

Never use ACS cards for FIPS authorization, because they do not support automatic recovery. Softcards or OCS must be members of the same Security World.

The following table describes the authorization recovery behavior of the nCipher Security World after a temporary outage.

Security World type	Protection/ Credential	Stand-alone system	Database cluster
FIPS level 2	Module	Recovers automatically	Recovers automatically
	Softcard	Recovers automatically	Recovers automatically
	OCS	Use OCS for credential authorization: (1) Use 1/N quorum. Same passphrase for all cards (2) Leave an OCS card in HSM slot. Recovers automatically.	Use OCS for credential authorization: (1) Use 1/N quorum. Same passphrase for all cards (2) Leave an OCS card in slot of every HSM in cluster. Recovers automatically.
FIPS level 3	Module	Use OCS for FIPS authorization (only): Leave an OCS card in HSM slot. Recovers automatically	Use OCS for FIPS authorization (only): Leave an OCS card in slot of every HSM in cluster. Recovers automatically
	Softcard	Use OCS for FIPS authorization (only): Leave an OCS card in HSM slot. Recovers automatically	Use OCS for FIPS authorization (only): Leave an OCS card in slot of every HSM in cluster. Recovers automatically
	OCS	Use OCS for both credential and FIPS authorization: (1) Use 1/N quorum. Same passphrase for all cards. (2) Leave an OCS card in HSM slot. Recovers automatically.	Use OCS for both credential and FIPS authorization: (1) Use 1/N quorum. Same passphrase for all cards. (2) Leave an OCS card in slot of every HSM in cluster. Recovers automatically.

If you are using an OCS to facilitate automatic recovery of the Entrust Security World:

- If you are using the OCS for credential authorization, all must be members of the same card set for the same credential, and the same passphrase must be assigned to every card in the set.
- If you are using the OCS for FIPS authorization purposes only, the quorum automatically defaults to 1/N, and (any) passphrase is ignored.

Authorization acquired through a persistent operator card does not automatically

reinstate itself after loss due to a temporary failure.

4.2. About the HSM credential

The protection methods available with the Entrust HSM are, in order of enhanced authentication:

- Module: Encryption keys are protected by a nCipher Security World protecting key in the HSM.
- Softcard: Encryption keys are protected by a named Softcard (software based) token key, a passphrase, and nCipher Security World protecting key in the HSM. You can have many different Softcards, but each is singular and works on its own.
- OCS: Encryption keys are protected by the presence of a named physical token (OCS smartcard), an OCS token key, a passphrase, and nCipher Security World protecting key in the HSM. OCS cards are usually part of a set of several OCS cards, or card set, and any member of the same card set protects the same encryption keys. You can have many different OCS card sets where each card set may protect different encryption keys.

The Softcard and OCS protection methods must be set up within the Entrust HSM before they can be used by an Oracle database. See your HSM *User Guide* for details. The module protection method can be used directly without any set up (other than the normal Entrust configuration). Setting up the Softcard or OCS includes creating and naming the token(s), with a passphrase (see the *User Guide* for your HSM).

Within SQL scripts as used by Oracle, identify the protection method using a `<credential>`. Choose the protection method you want to use for `<credential>` from the table below.

Protection Type	Credential or <credential>
Module protection	`<module-passphrase>`. In this case the passphrase is an access mechanism for Oracle, and is not used by the nShield HSM
Softcard protection	`<softcard-passphrase>|<softcard-name>`
OCS protection	`<OCScard-passphrase>|<OCScard-name>`

Oracle documentation gives the ordering `<credential-name>|<credential-passphrase>`. However, tests showed that the ordering `<credential-passphrase>|<credential-name>`` works.

Oracle SQL uses the separator symbol **|** or **:** to divide the `<credential-passphrase>` and

<credentialname>. Hence the total Oracle SQL string for a credential comprises:

- Module protection: <passphrase>
- Softcard or OCS card protection: <credential-passphrase> + <separator> + <credential-name>.

In the nCipher Security World, Entrust recommends the following restrictions on token names, or credentialname:

- Maximum length of 254 characters.
- ASCII 7-bit characters only, restricted to:

  A-Z, a-z, 0-9, $ - _ (no white space).

In the nCipher Security World, Entrust places the following restrictions on passphrases, or credential- passphrases:

- Maximum length of 254 characters.
- ASCII 7-bit characters only:

  A-Z, a-z, 0-9, ! @ # $ % ^ & * - _ + = [ ] { } | \ : ' , . ? / ` ~ " < > ( ) ; (no white space).

However, the Oracle SQL interface imposes further restrictions on top of the nCipher restrictions for what can comprise the string <credential-passphrase> + <separator> + <credential-name>, as follows:

- The total string length, including separator, can be no more than 30 characters. This leaves 29 characters for the <credential-passphrase> + <credential-name>.
- The symbols **| : "** and **'** cannot be used within the <credential-passphrase> or <credential-name>.

From the Oracle side, if:

- N is the length of the credential name.
- P is the length of the credential passphrase, then 2 ⇐ (N+P) ⇐ 29, where 1 ⇐ N ⇐ 28, and 1 ⇐ P ⇐ 28, assuming a minimum of one character for passphrase and name.

  Permitted symbols are:

- <credential-passphrase>:

  A-Z, a-z, 0-9, ! @ # $ % ^ & * - _ + = [ ] { } \ , . ? / ~ < > ( ) ; (no white space)

- <credential-name>:

  A-Z, a-z, 0-9, $ - _ (no white space).

Use a passphrase of sufficient length to meet your current security requirements.

Oracle (wallet manager) states “Passwords must have a minimum length of eight characters and contain alphabetic characters combined with numbers or special characters".

When you are using a Softcard or OCS credential, an SQL script that uses the credential must get the `<credential-passphrase>` and `<credential-name>` exactly correct. If there is a mistake, then the entire credential string may be misinterpreted as a `<module-passphrase>`. Your encryption keys are then placed under module protection rather than the Softcard or OCS card protection you intended. For this reason, after creating encryption keys or rekeying, then immediately use the nCipher `rocs` utility to check the keys you have just created are under the expected credential or protection method.

In the examples shown in this guide, credentials may be given descriptive names to make it clear what they are used for, such as `<keystore-credential>`. In practice, replace the descriptive names with the actual credential passphrases and names you are using. If you want to change the passphrase for Softcards or OCS cards, you must change the passphrase for the token in the nCipher Security World first, followed by updating the change to the database. For module protection you need only change the passphrase as seen by the database.

If you are using a FIPS 140-2 Level 3 Security World:

- To change the passphrase of a Softcard, or create a new Softcard, you require either authorization using ACS cards, or an OCS authorizing card.
- To change the passphrase of an OCS card, or create a new OCS card, you require authorization using ACS cards.

You can change the protection method or credential in one of the following ways:

- Continue using the same protection method and token, but change the associated passphrase. There is no token for module protection, but you can change the passphrase. In this case, after the passphrase is altered, TDE continues working using the new passphrase, because the protected TDE encryption keys remain the same.
- Continue using the same protection method, but change the token and passphrase. In this case, you have two options:
    1. If you are not transferring encryption keys from the previous token to the new token, you can no longer continue using TDE as protected by the previous token’s keys. You will only be able to use TDE encryption keys shielded under the newly activated credential.

2. If you are transferring encryption keys from the previous token to the new token, you can continue using TDE as protected by the previous token's keys. However, you can only transfer keys from different Softcards, or different OCS cards. You cannot transfer keys between Softcards and OCS cards.

- Change the protection method and associated credential with passphrase. In this case, you cannot transfer encryption keys between the different protection methods. You can only use TDE encryption keys shielded under the new protection method and credential.

4.2.1. Change passphrase only

1. To change a passphrase only, complete the following instructions:
   - Non-multitenant:

     CONNECT TESTER@DB

     ```
     -- If Database is not open already
     ALTER DATABASE OPEN;

     -- Pre-11.2.0.1.0
     ALTER SYSTEM SET ENCRYPTION WALLET CLOSE;

     -- 11.2.0.1.0 onward
     ALTER SYSTEM SET ENCRYPTION WALLET CLOSE IDENTIFIED BY "<credential>";
     ```

   - Multitenant:

     CONNECT C##TESTER@CDB<n>

     ```
     -- If Database is not open already
     ALTER PLUGGABLE DATABASE ALL OPEN;

     ADMINISTER KEY MANAGEMENT SET KEYSTORE CLOSE IDENTIFIED BY "<old-credential>" CONTAINER=ALL;
     ```

2. At this point:
   - If you are using module protection, skip to the next SQL statements.
   - If you are using Softcard protection, refer to the *User Guide* for your HSM for instructions on how to change the Softcard passphrase using the `ppmk` utility.
   - If you are using OCS protection, refer to the the *User Guide* for your HSM for instructions on how to change the OCS passphrase using the `cardpp` utility. If you are using OCS cards, all OCS cards within the same (1/N) card set must be altered to share the exact same passphrase.
3. Bounce the database:
   - Non-multitenant:

     CONNECT TESTER@DB

```
-- If Database is not open already
ALTER DATABASE OPEN;

ALTER SYSTEM SET ENCRYPTION WALLET OPEN IDENTIFIED BY "<new-credential>";
```

- Multitenant only:

  CONNECT C##TESTER@CDB<n>

```
-- If Database is not open already
ALTER PLUGGABLE DATABASE ALL OPEN;

ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN IDENTIFIED BY "<newcredential>" CONTAINER=ALL;
```

4.3. Change token with associated passphrase but keep same protection method

This does not apply to module protection.

1. To change a token with passphrase for the same protection method, complete the following instructions:
   - Non-multitenant:

     CONNECT TESTER@DB

```
-- If Database is not open already
ALTER DATABASE OPEN;

-- Pre-11.2.0.1.0
ALTER SYSTEM SET ENCRYPTION WALLET CLOSE;

-- 11.2.0.1.0 onward
ALTER SYSTEM SET ENCRYPTION WALLET CLOSE IDENTIFIED BY “<credential>”;
```

   - Multitenant:

     CONNECT C##TESTER@CDB<n>

```
-- If Database is not open already
ALTER PLUGGABLE DATABASE ALL OPEN;

ADMINISTER KEY MANAGEMENT SET KEYSTORE CLOSE IDENTIFIED BY "<old-token-credential>" CONTAINER=ALL;
```

2. At this point:
   - If you do not want to transfer TDE encryption keys from the previous token to the new token, skip to the next SQL statements. If you are using an OCS card set (1/N), all OCS cards within the new card set must share the exact same passphrase.

If you do want to transfer TDE encryption keys from the previous token to the new token, refer to your HSM *User Guide* for instructions on how to transfer the keys using the `rocs` utility.

It is recommended to back up your Security World data before transferring keys between tokens. See the *User Guide* for your HSM.

To transfer keys using the `rocs` utility, you will need your Security World ACS cards to authorize transfer of keys between tokens. You can only transfer encryption keys between Softcards, or else between OCS cards, but not between Softcards and OCS cards. If transferring keys to another OCS card set (1/N), all OCS cards within the target card set must share the exact same passphrase.

3. Bounce the database:
   - Non-multitenant:

     CONNECT TESTER@DB

     ```
     -- If Database is not open already
     ALTER DATABASE OPEN;

     ALTER SYSTEM SET ENCRYPTION WALLET OPEN IDENTIFIED BY "<new-tokencredential>";
     ```

   - Multitenant:

     CONNECT C##TESTER@CDB<n>

     ```
     -- If Database is not open already
     ALTER PLUGGABLE DATABASE ALL OPEN;

     ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN IDENTIFIED BY "<new-tokencredential>" CONTAINER=ALL;
     ```

4.3.1. Change protection method

1. To change the protection method, complete the following instructions:
   - Non-multitenant:

     CONNECT TESTER@DB

```
-- If Database is not open already
ALTER DATABASE OPEN;

-- Pre-11.2.0.1.0
ALTER SYSTEM SET ENCRYPTION WALLET CLOSE;

-- 11.2.0.1.0 onward
ALTER SYSTEM SET ENCRYPTION WALLET CLOSE IDENTIFIED BY “<credential>”;
```

- Multitenant:

  CONNECT C##TESTER@CDB<n>

```
-- If Database is not open already
ALTER PLUGGABLE DATABASE ALL OPEN;
ADMINISTER KEY MANAGEMENT SET KEYSTORE CLOSE IDENTIFIED BY "<old-protection-credential>" CONTAINER=ALL;
```

  If you are using OCS cards, all OCS cards within the same (1/N) card set must share the exact same passphrase.

2. Bounce the database:
   - Non-multitenant:

     CONNECT TESTER@DB

```
-- If Database is not open already
ALTER DATABASE OPEN;

ALTER SYSTEM SET ENCRYPTION WALLET OPEN IDENTIFIED BY "<new-protection-credential>";
```

   - Multitenant only:

     CONNECT C##TESTER@CDB<n>

```
-- If Database is not open already
ALTER PLUGGABLE DATABASE ALL OPEN;

ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN IDENTIFIED BY "<new-protectioncredential>" CONTAINER=ALL;
```

4.4. Latency issues

It is beyond the scope of this guide to deal with specific solutions to latency issues, and these will only be discussed in general terms.

When you are using an Oracle database, the nCipher Security World provides and protects the master encryption keys (wrapping keys) that are used to wrap Oracle symmetric keys that are in turn used for tablespace or table column encryption. The Oracle symmetric keys are stored as part of the database itself, although protected by the wrapping key.

In the context of this guide, encrypted data will be taken to include the symmetric key(s) that are stored as part of an Oracle database, as well as the encrypted data itself. Master encryption keys will be taken to be the (wrapping) keys stored by the nCipher Security World. Latency issues may occur when there is a mismatch between the encrypted data and the (correct) master encryption keys, due to a time lag in an update of either. This should only be a problem where there are multiple clients using the same database and encryption keys. In this case, when data or master encryption keys are updated on one client, the changes must be distributed before use by the other clients. Otherwise synchronization problems may occur. Note that the client that initiates the changes should suffer no synchronization problems.

Typically, these issues are more complex to resolve for a large and geographically distributed database system, rather than a small or localized system. It is the job of the system administration to ensure that encrypted data is synchronized with the appropriate master encryption keys at any particular time. Furthermore, is not within the control of the nCipher software if encrypted data does not match (the correct) master encryption keys in the Security World because of a time lag in updating the database.

Where there may be a time lag in updating master encryption keys in the Security World to match encrypted data, this may be due to the following:

- Time lag in distributing new or updated master encryption keys to a Security World, or between different copies of the same Security World, after a key rotation or rekey.
- After new or updated master keys have been successfully distributed to the Security World, then a lag in making a nCipher hardserver instance recognize the new master keys.

4.4.1. Storage and distribution of updated master keys

4.4.1.1. Common storage of master encryption keys

Entrust recommends configurations where the Security World data is held in common storage between clients that require use of the same master encryption keys (if possible).

If common storage of the master encryption keys is being used, then there may be a short time delay before newly created keys are successfully copied to the common store. After this, there may be a further short time delay before a client is able to access the keys from the common store. The time period a client may not be able to access the updated keys is likely to be very short, but may increase if the client is geographically distant from the common store and communication delays accumulate. Note that if you are using a common store, the master keys are implicitly updated for the use of all clients, and there is no need to trigger any other update mechanism.

Common key storage implies:

- Key update is implicit and simple (as there is only one store).
- Keeps time delays short, thereby minimizing any problems synchronizing keys with data.
- It is essential the common store is backed up frequently, as otherwise it is the only copy of the encryption keys.

4.4.1.2. Local storage of master encryption keys

If each client is using its own local copy of the Security World, then after an update of the master keys is initiated on any client, the updated keys must be distributed in a timely manner to the local Security Worlds of every other client. To achieve this, there must be some explicit update mechanism in order to recognize when an update is required in the first place, and then trigger the key distribution process.

Clearly, if this was done manually, it is likely to be a slow process. If it is done automatically, recognizing when a rekey occurs should not be difficult on the client that initiates it, and triggering the update should not therefore be a problem. Even so, for a configuration that uses dispersed local copies of the Security World, mechanisms to distribute the updated keys are likely to be slower and more difficult to implement than for the common key storage case. This makes the timely synchronization of the master keys with the data more problematic.

Entrust provide the utilities `rfs-setup`/`rfs-sync` (gang-client) that can provide limited facilities to distribute keys between different clients, although you must use an RFS for intermediate key storage. However, these utilities were originally designed for manual operation. Clearly, these utilities can be incorporated into automated scripts customized for your particular configuration. But elaborating this into an automated system to distribute your keys without synchronization problems is a task for your system development team. Further information about nCipher `rfs-setup`/`rfs-sync` utilities can be found in the *User Guide* for your HSM.

An alternative for key distribution is the UNIX `rsync` utility. However, it is beyond the scope of this guide to discuss how this may be used.

If you require further assistance for distributed key update arrangements, contact Entrust Support.

Local key storage and distribution implies:

- An explicit update mechanism that may be complex to automate.
- Greater difficulty in keeping distribution time delays short, increasing any problems in synchronizing keys with data.

- There are multiple copies of the Security World, making the loss of any one copy less significant than may be the case with common storage.

4.4.1.3. Making a hardserver instance recognize new master keys

In a configuration with multiple clients sharing the same encryption keys, if a rekey is performed, the new keys should be immediately available and usable on the client that performed the rekey. However, for the other clients, after the new keys have been made available in their Security World folder, for the new keys to become usable to the local hardserver instance, you have a choice of the following options (this applies for both shared and local key storage):

1. In the nCipher `cknfastrc` file for each client, insert the following:

   ```
   CKNFAST_ASSUME_SINGLE_PROCESS=0
   ```

   This will ensure the Security World folder is scanned for the latest keys whenever a key is required, and avoids key caching. However, with this option the Security World will be scanned every time a key is required, even if no new keys have been added to the Security World. If there are many keys this may take a significant time. Additionally, as it will be repeated every time a key is needed, it may slow down overall operations. However, use of this option should not require downtime for the key update.

2. For each client that did not initiate a rekey, all applications/users that were using encryption keys on the database should be reconnected. A new connection will force a scan of the Security World that will pick up new keys. But in this case, it is a single scan for that connection, and will NOT be repeated every time a key is required. If you have many keys, encrypted database operations will be temporarily hindered only on the occasion you need to make a reconnection to update your master keys. Use of this option may imply temporary downtime while reconnections are made after a key update. But if you routinely make new connections on your system per transaction, this should be hardly noticeable.

4.4.1.4. Other considerations

Even if a client is not able to access the required master keys for a short period, this is not necessarily a serious problem. The Oracle database should be able to recover gracefully if unable to obtain the correct master key(s). It should be possible to program the database to rollback failed transactions and make several attempts to repeat the transaction, until some expiry point is reached.

If the delay in the update of the master keys is short, then repeated attempts at the transaction should eventually succeed when the master key update is complete. If it is

not possible to do this within the Oracle database itself, then it should be possible to do something similar in the application code that is using the database.

If you are using the common shared storage, it is expected that any lag in updating the master keys will be short enough that either:

- The Oracle database will not be affected.
- The Oracle database will cope gracefully, and subsequently recover automatically as described above, as and when the update completes.

If delays in updating the master keys exceed the limits of what the Oracle database or application can cope with gracefully, then it may be necessary to halt encryption transactions temporarily while a master key rotation is performed.

Entrust strongly recommends you test your solutions in a safe environment before transferring to a production environment.

4.5. How Oracle works with the Entrust HSM

Before using the Entrust HSM, either a new Security World must be created using the HSM, or a previously created Security World must be loaded onto the HSM. For more information, see the *User Guide* for your HSM.

The Security World is stored in a folder on your host server(s) and holds the database encryption keys, and associated credential files, that are to be protected. All data in the Security World folder is automatically encrypted and is useless to anyone without the authorized access and decryption mechanisms. When encryption keys are to be used, they are loaded into the physically protected environment of the HSM where they may be securely decrypted for use. Encryption keys protected by an HSM are never available in plaintext outside the boundary of the HSM. Legitimate use of the encryption keys is authorized and protected as described below.

If you are creating a new Security World, you must create an Administrator Card Set (ACS). An ACS is a set of physical smartcard(s) that must be used to create a Security World. When the Security World has been created, the ACS is used to secure the higher administrative functions of the Security World. Without a quorum of ACS cards, you cannot create or load a Security World onto an HSM, or alter it. Each ACS card can be issued with a unique passphrase and is specific to the Security World. When the Security World is created, you must stipulate a minimum number of cards, known as a quorum, required to load the Security World onto an HSM at any later time. However, the number of cards in the set should exceed the quorum, so that spares are available in case of failures or loss of card. An encrypted copy of the created Security World is stored in a folder on the host server(s).

If you are loading an existing Security World onto an HSM, you need access to a folder

holding the Security World, and a quorum of the same ACS cards, and associated passphrase(s), that were used to create the Security World.

After the Security World has been created or loaded onto the HSM, a suitable HSM protection method may be prepared, or resumed if it was already present in an existing Security World. The protection method enables authorized access to the encryption keys assigned to it. The following protection methods are available, in order of increasing authentication requirements:

- Module protection: Oracle master encryption keys are protected by a Security World protecting key.
- Softcard protection: Oracle master encryption keys are protected by a (singular) named software token key, a passphrase, and Security World protecting key.
- Operator Card Set (OCS) protection: Oracle master encryption keys are protected by the presence of a set of named physical token(s) or smartcard(s), an OCS token key, and Security World protecting key. An OCS smartcard set is similar to the ACS card set in that it must stipulate a quorum of cards to authorize permission to use its protection. The number of cards in the set should exceed the number of HSMs that may be sharing the same Security World so that spares are available in case of failure. The card set should have a unique name that covers all cards in the set. In typical use with Oracle, all OCS cards in the same set should have the same passphrase, and the quorum is one.

For instructions to set up these protection methods, see the *User Guide* for your HSM.

If you have loaded an existing Security World onto the HSM and will be using an OCS card set that it already contains, you must use the same physical OCS cards and associated passphrase(s) that were originally created in that Security World. Similarly, for Softcard or module protection, you will need the original passphrase(s).

In this *Integration Guide*, the word credential is used for a passphrase, or the combination of a passphrase and a named token (OCS or Softcard). Before an Oracle database can make use of the facilities offered by the nShield HSM, it must have access to the nCipher library file `libcknfast.so` which is installed as described in this guide. This is vital, as without access to the nCipher library file, the Oracle database and nShield HSM or nCipher software cannot communicate. Once successful communication is established between the Oracle database and nShield HSM, the Oracle database can gain access to the HSM by use of a credential incorporated into an SQL script. When it is set up with a credential, the Oracle database can proceed to create and assign encryption keys to that credential if no encryption keys yet exist, and encrypt or decrypt data using the encryption keys protected by that credential.

A protection method or credential is uniquely associated with the Security World where it was created and cannot be used with any other Security World. It should also be uniquely associated with the encrypted database(s) it is protecting. An encrypted

database cannot be decrypted without access to the same master keys that protect it (likely to be an asymmetric pair). If you use OCS protection, the Oracle database must use the correct OCS card name and associated passphrase in its SQL scripts to access the encryption keys assigned to the OCS. Likewise, if you use a Softcard, the Oracle database must use the correct Softcard name and associated passphrase in its SQL scripts to access the encryption keys assigned to the Softcard.

If you use module protection, a passphrase is required for the Oracle database access mechanisms only. The Oracle module protection passphrase does not have a reference or counterpart in the nShield HSM. This means that a user who is able to access keys directly in the HSM is able to access module protected keys for any database without requiring the Oracle passphrase. This does not apply for Softcard or OCS protection.

Use of the HSM credentials and associated SQL scripts that open up access to the encrypted data should be strictly limited to authorized persons. However, the system can be set up so that approved clients can retrieve the encrypted data that is automatically decrypted when it leaves the database. Approved database users do not need the HSM credentials and associated SQL scripts to do this. They can continue to use the database as normal. Encryption should be invisible to them in most circumstances.

If you first use an Oracle software keystore to protect the master encryption keys, but later want to switch to an HSM, the encryption facilities can be migrated to the HSM. Also, encryption facilities can be migrated from an HSM back to an Oracle software keystore. During migration, fresh master key(s) are created in the HSM or software keystore, and the subsidiary keys that are being protected are re-encrypted with the new master key(s). Legacy keys remain in the software keystore or HSM where they were created, and should be (securely) retained in case they were used for past backups or other legacy data. For more information on key migration, see the Oracle documentation.

For loading or failover, you can use more than one HSM in the same system. The HSMs must share the Security World, and operate together to provide the same functions as a single HSM.

There is some performance degradation when Transparent Data Encryption (TDE) is used. The impact depends on the types of transactions you typically perform. Using the Security World software and the HSM usually have a negligible impact on TDE performance. You should test your Oracle and HSM configuration in a realistic test environment before committing to a production environment.

All nShield HSMs are FIPS certified to 140-2 Level 3, meaning that they are tamper evident and tamper resistant. nShield Connects are also tamper responsive, if an attempt to open the nShield Connect body is detected, all stored HSM encryption key data is deleted.

The encryption facilities described in this document are designed only to protect data at rest. TDE encrypts data while stored on disk, but once the data is retrieved to working memory, it is in plaintext and can be read by anyone able to access it. Decrypted data in transit between a database server and client should be independently encrypted to ensure security during data transfer. Security World data is inherently encrypted. There should be minimal security risk in transmitting this data over open networks. Similarly, encrypted database contents should be minimally at risk if transmitted over open networks.