Oracle 19C Transparent Data Encryption

nShield® HSM Integration Guide

2026-06-05

© Entrust Corporation. All rights reserved.

Table of Contents

Chapter 1. Introduction

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

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

The integrated Oracle and Entrust solution has been tested with Oracle TDE for tablespace encryption, column encryption, or both concurrently. Entrust nShield HSMs are certified to FIPS 140 Level 3, delivering a high-grade level of security assurance. They protect sensitive encryption keys and offload encryption and key management operations from the database server.

1.1. Using this guide

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

- An overview of how the Oracle database software and the Entrust Security World software with HSMs 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 can be used.
- Troubleshooting advice.

This guide assumes that:

- You have a good working knowledge of Oracle database technology.
- You have read the Security World and HSM documentation, and are familiar with the documentation and setup process for Oracle database TDE.
- You already have an Oracle database installed.

After the Entrust Security World software and HSM are installed and configured, no

additional software is required; only minor configuration changes are needed.

This guide cannot anticipate every configuration requirement. The examples provided are not exhaustive and do not necessarily show the simplest or most efficient way to achieve a given result. Use them as a starting point for integrating the Entrust HSM with an Oracle database, and adapt them to your own circumstances.

Entrust strongly recommends that you thoroughly validate your solutions in a safe test environment before committing them to production. If you require additional help in setting up your system, contact Entrust Support.

Entrust accepts no responsibility for:

- Loss of data or services incurred through use of the examples in this guide, or through errors in this guide.
- Information in this guide that becomes obsolete due to changes or upgrades to Oracle products.

1.2. Product configuration

Entrust has successfully tested nShield HSM integration in the following configuration:

OS Version	Kernel	Oracle Version
Red Hat Enterprise Linux 8	4.18.0-553.125.1.el8_10.x86_64	Oracle Database 19c - 19.31.0.0.0

1.2.1. Supported nShield hardware and software versions

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

HSM	Security World Software	Firmware	Image	OCS	Softcard	Module	FIPS Level 3
nShield Connect	13.6.16	12.72.4 (FIPS 140-2 certified)	13.6.16	✓	✓	✓	✓
nShield 5c	13.6.16	13.4.5 (FIPS 140-3 certified)	13.6.16	✓	✓	✓	✓

1.3. Conventions used in this document

1.3.1. Multitenant and non-multitenant

This *Integration Guide* covers both non-multitenant and multitenant databases, and uses the terms appropriate to the database type under discussion:

- Non-multitenant databases run on Oracle version 11g or earlier. Multitenant databases were introduced in Oracle version 12c.
- Non-multitenant database software can only create and use non-multitenant databases. When non-multitenant databases are the subject matter, the non-multitenant terminology and SQL shown below applies.
- Database software that supports multitenant databases may also support non-multitenant databases (pre-21c). In this case, when a non-multitenant mode is the subject matter, the non-multitenant terminology and SQL shown below applies. When a multitenant mode is the subject matter, the multitenant terminology and SQL applies.

The following terminology is used:

- Non-multitenant (non-container):
  - Oracle software-based encryption key repository: `ALTER SYSTEM SET ENCRYPTION ...`
- Multitenant (container):
  - Oracle software-based encryption key repository: software keystore.
  - SQL preamble for encryption-related commands: `ADMINISTER KEY MANAGEMENT`, and so on.

Where terminology applies equally to a software wallet or keystore, this guide uses *software keystore* as the default term for both.

1.3.2. Database connections

To access a database, you must be a user with the correct permissions, and you must also have the privileges required to perform the operations you intend to carry out while connected. Your system administrator can create users and grant suitable permissions and privileges according to your organization's security policies.

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

The following database users and database identifiers are used in the examples in this

guide.

- <database-user>. This guide uses one of the following users to connect to databases:
  - sysdba: Oracle's standard sysdba user.
  - system: Oracle's standard system user.
  - Non-multitenant:
    - TESTER, as a local user.
  - Multitenant:
    - C##TESTER, as a common user for the 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 uses one of the following database identifiers when connecting:
  - 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 are:

    - CDB<n>, to connect to 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, a connection implies that you must alter the session if you are not already connected to the required container. For example:

- Example 1:

```
CONNECT C##TESTER@CDB<n>
```

If you are not already connected to `CDB<n>`, alter the session:

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

- Example 2:

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

If you are not already connected to `CDB<n>PDB<k>`, 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@CDB1$ROOT`
- `sqlplus CDB1PDB1TESTER/Tester@//localhost:1521/CDB1PDB1.interop.com`

1.3.3. Key migration and legacy keys

Encryption master keys can be migrated from an existing Oracle keystore to an Entrust HSM, or vice versa. In this guide, the term *key migration* means that the responsibility for holding the master keys is being transferred between key protectors. The encryption keys themselves are not copied or imported between a software keystore and the HSM Security World. Instead, fresh master keys are created within the software keystore or HSM that is to become the new key protector, and any subsidiary keys being protected are re-encrypted using the new master keys. From that point on, any new master keys are created in the current key protector you have migrated to.

During rekey, the previous master keys (the *legacy keys*) remain in the software keystore or HSM where they were created. After a key migration, you can retain access to the legacy keys in the software keystore or HSM you have migrated away from by setting its passphrase to the same value as the current key protector's. This allows both to be open at the same time, providing access to the encryption keys they each contain. If you do not do this, you can only access keys in the current key protector. When you are using both a software keystore and an HSM at the same time, the current key protector is referred to as the *primary*.

1.4. Overview

Transparent Data Encryption (TDE) encrypts an entire database without requiring changes

to existing queries and applications. A database encrypted with TDE is automatically decrypted when it is loaded from disk storage into memory, so clients can query the database within the server environment without having to perform any decryption operations themselves. The database is re-encrypted when it is written back to disk. Note that with TDE, data is not protected by encryption while it is in memory.

The encryption keys used to encrypt the database are typically held as part of the database, but they are themselves encrypted with a master encryption key. Using an Entrust HSM allows the master encryption keys to be kept physically separate from the database they protect, and provides a hardware-protected boundary that encryption keys can never leave in plaintext. The encryption keys are stored in a Security World folder that is also encrypted, making it useless to anyone without the authorized means to access them. The Security World folder can be easily backed up or transferred to other legitimate clients that use the authorized mechanisms to access the encryption keys.

Entrust recommends that you allow only unprivileged connections unless you are performing administrative tasks.

Other benefits of using the nShield HSM include:

- Centralized storage of keys from across the enterprise, for easier management.
- Key retention: rotate keys while keeping the previous ones.
- FIPS and Common Criteria compliance.

Chapter 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 that you have an agreed organizational Certificate Practice Statement and a Security Policy or Procedure in place covering administration of the HSM. In particular, these documents should cover the following aspects of HSM administration:

- Whether the Security World must comply with FIPS 140 Level 3 or Common Criteria restrictions.
    - If you want to use a FIPS 140 Level 3 Security World, 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 card set can be used for all database instances.
    - If you want to use OCS protection, the 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 those 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, decide on the number of Operator Cards in the OCS card set. K/N functionality is not currently supported, so you must create 1/N OCS card sets. The number of OCS cards in a card set must be at least equal to the number of HSMs in your configuration, plus extras in case of card loss or failure.
- 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.
- A policy for managing the passphrases for your:
    - ACS
    - Module protection

    - Softcard protection
    - OCS protection

      For information on passphrases, see About the HSM credential.

- A policy for managing the physical security of the smartcards used for the ACS and OCS, and their deployment to authorized users.

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

This guide assumes that the Oracle database software, and at least one Oracle database, are already installed on your system. Ensure that any required patches have been applied.

To integrate an Oracle database with a Entrust HSM, perform the following steps:

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

The details of your installation and configuration depend on 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 setting up your system, see the following appendixes:

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

2.2. Basic setup

1. Install the Entrust Security World software on each client in accordance with its accompanying documentation. If you are using nShield network-attached HSMs 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 keys, set the

following PKCS #11 environment variables:

- For OCS or Softcard key protection and for HSM load sharing:

```
CKNFAST_LOADSHARING=1
CKNFAST_OVERRIDE_SECURITY_ASSURANCES=all
```

- For module key protection:

```
CKNFAST_FAKE_ACCELERATOR_LOGIN=1
CKNFAST_OVERRIDE_SECURITY_ASSURANCES=all
```

For more information, see the description of the PKCS #11 library environment variables in the PKCS #11 Reference Guide for nShield Security World.

3. If you are using nShield network-attached HSMs, configure them to operate with your selected RFS and clients as described in your HSM documentation. Typically the clients are the host servers on which your Oracle database is running.
4. Configure the Oracle PKCS #11 library folder to use the Entrust PKCS #11 API.

   After creating the Oracle database:

   a. Create the following directory path for the Entrust API library as the `oracle` user. Set ownership and permissions on the directory to `owner=oracle; group=oinstall; permissions=775`:

```
mkdir -p $ORACLE_BASE/extapi/64/hsm/nCipher/13.6.16
chown oracle $ORACLE_BASE/extapi/64/hsm/nCipher/13.6.16
chgrp oinstall $ORACLE_BASE/extapi/64/hsm/nCipher/13.6.16
chmod 775 $ORACLE_BASE/extapi/64/hsm/nCipher/13.6.16
```

   b. Copy the PKCS #11 library into the directory as the `oracle` user:

```
cp /opt/nfast/toolkits/pkcs11/libcknfast.so $ORACLE_BASE/extapi/64/hsm/nCipher/13.6.16
```

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, the two systems cannot 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 an nShield network-attached HSM. 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 on all clients and on the RFS, and that it is loaded onto each nShield network-attached HSM used in the configuration.
2. Check the Security World on the 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 network-attached HSM:
     - Front panel: **MENU > Security World mgmt. > Display World Info**.

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

2.2.2. Prepare protection method

1. If your Security World does not already contain the required protection method, proceed as follows:
   - If you want to use module protection, no action is required at this point. You will complete the required module-protection steps later in this integration process.
   - If you want to use Softcard protection, create the required number of Softcards, each with its own passphrase.
   - If you want to use 1/N OCS card set protection, create the required number of card sets now, using the 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 Level 3 environment, 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, create one for this purpose.

2.3. Configuring Oracle database software to use the Entrust HSM

This section assumes that:

- You have followed the setup and configuration instructions in this guide, including:
  - 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 operate with 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 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 the HSM**: One or more database instances are not using TDE encryption, and you want to start using TDE encryption for at least one database, with the Entrust HSM.

Before attempting key migration, see Key migration and legacy keys.

The SQL commands used later in this document might:

- Require more than one user with suitable database privileges to make specific database connections and run the SQL commands in the sequence shown. Follow the connection sequence shown to run the SQL successfully on your target system. See Database connections. Your system administrator should have sufficient knowledge to create users and assign privileges according to your organization's security policies.
- Need to be run as a particular user. If you are instructed to make a connection as a particular user, continue with that connection until instructed otherwise.
- Use `<credential>` to denote your chosen protection method. Once a protection method has been invoked, you must continue with the same protection method unless you decide to change it as described in About the HSM credential.

When using a Softcard or OCS card, the `<credential>` is a string containing the passphrase and card name of the card being used, in the form `<credential-passphrase>|<credential-name>`.

In SQL, the credential used to open a keystore must match the credential used to create an encryption key.

Whenever you have finished migrating or creating encryption keys in an HSM, Entrust recommends that you back up your Security World data.

> See the *User Guide* for your HSM.

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

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

2.4.1. Non-multitenant

This section assumes the database is open.

- To open or close the wallet:

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

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

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

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

  The `[ENCRYPTION]` clause is optional.

2.4.2. Multitenant

This section assumes that the respective CDB and PDB databases are open.

- To open or close the 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 or close the keystore for the container (CDB) and all the 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;
  ```

  To close all keystores, use:

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

- To open or close the keystore for a single PDB, you must use the same credential as the containing CDB:

  CONNECT CDB<n>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>";
  ```

2.4.3. Issues closing keystores

During migration from a software wallet to an HSM keystore, you might experience issues closing the keystore. To resolve this, disable the auto-login keystore so that all keystores can be closed:

```
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, that credential is activated. You can then create master encryption keys, or use any existing master encryption keys, 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 use different credentials simultaneously for different database instances on the same host server. For a container database, only the 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.

2.5.1. 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 that you back up your software wallet as a separate operation before attempting migration to the HSM. Keep the backup folder in a safe place, separated from the associated database files. Only authorized users should be able to access the backup folder.

Repeat the following procedure for each database software wallet you want to migrate. Each 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, 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 that the `WALLET_ROOT` parameter has already been set for Oracle keystore use.

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

   `CONNECT sysdba@DB`

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

2. Migrate from the keystore to the 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.

2.5.2. Migrating from software keystore to HSM (multitenant)

The following procedure applies when the target database is multitenant and you are already using a software keystore 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 you want to migrate. Each

container database (CDB) 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), 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 "<keystore-passphrase>";
   ```

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

   `CONNECT sysdba@CDB1$ROOT`

   ```
   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@CDB1$ROOT`

   ```
   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 the HSM:

   `CONNECT sysdba@CDB1$ROOT`

   ```
   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.

2.5.3. Create master keys directly in an HSM for a 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 a 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, 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.

2.5.3.1. Use the WALLET_ROOT and TDE_CONFIGURATION parameters

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

   `CONNECT sysdba@DB`

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

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

2.5.3.2. Create the encryption keys

1. Select the protection method (credential) that you require, 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 have created the master encryption keys in the HSM, proceed to encrypt your database using tablespace encryption, column encryption, or both.

2.5.4. Create master keys directly in an HSM for a multitenant database

The following procedure applies when the target database is multitenant and there is no pre-existing software keystore. If your target database is non-multitenant, see Create master keys directly in an HSM for a 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), 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. Once the CDB master key has been created, you can choose to create master keys for all the PDBs it contains in a single operation, or for each PDB individually.

The PDBs must use the same protection credential as the CDB.

2.5.4.1. Use the WALLET_ROOT and TDE_CONFIGURATION parameters

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

   `CONNECT sysdba@CDB1$ROOT`

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

2. Bounce the database after setting 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 the `TDE_CONFIGURATION` parameter.

2.5.4.2. Create the CDB and all PDB master keys in a single operation

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

   `CONNECT C##TESTER@CDB<n>`

   ```
   ALTER PLUGGABLE DATABASE ALL OPEN;
   ```

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

2. Activate the master keys for the CDB and all its PDBs in a single 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.

2.5.4.3. 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, and run the SQL:

   ```
   --This activates the credential if it isn't already active
   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 to create a single PDB master key for any PDB you select.

2. Create a single PDB master key:

   `CONNECT CDB<n>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>";

--Create the master key for the PDB you are 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.

2.6. Rekeying or key rotation

After you have established your HSM as the primary protector for your master encryption keys, you may want to periodically replace the keys (*rekey*) for security reasons. You can do so by following the instructions below.

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

2.6.1. Rekey when sharing keys between clients

When encryption keys are shared or distributed between clients, either a common shared Security World folder or local client copies of the Security World folder are 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, first make sure that the new keys are available in the Security World folder it is using. Once the new keys are available, you can make the client hardserver instance recognize and use them in either of the following ways:

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

   ```
   CKNFAST_ASSUME_SINGLE_PROCESS=0
   ```

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

These actions cause the available keys to be scanned by the client's hardserver instance, after which any new keys are recognized and made usable. See Latency issues to understand the full consequences of these options.

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

operation.

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

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

2.6.2. Rekey for a non-multitenant database

The following instructions assume that the HSM (wallet) is already open.

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

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

2.6.3. Rekey for a multitenant database with CDB and all PDBs in a single operation

`CONNECT C##TESTER@CDB<n>`

The following instructions assume that the required CDB has started, and that the required PDBs and HSM (keystore) are already open.

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

2.6.4. Rekey for a multitenant database with the CDB only

The following instructions assume that the required CDB has started and the HSM (keystore) is already open.

`CONNECT C##TESTER@CDB<n>`

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

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

The following instructions assume that the required CDB has started, and that the required PDB and HSM (keystore) are already open.

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

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

Chapter 3. Troubleshooting

Oracle error messages can sometimes describe symptoms rather than the root cause. If you see an unfamiliar error, consult Oracle documentation and trusted support resources before attempting to resolve it. If you are still unable to resolve the error, contact Oracle support.

If you edit an Oracle configuration file, use a plain text editor running on the host. Do not copy and paste file contents using a rich-text editor, because it may insert hidden characters that are difficult to detect and can prevent the file from working. Entrust also recommends that you avoid copying files (including library files) onto a UNIX host via a Windows intermediary.

3.1. An SQL command produces no output, or unexpected output or an error

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

3.2. After changing a configuration file, database behavior does not change

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 the correct permissions to use the `/opt/oracle/extapi/…` directory.
2. Check that you are using a library built for the correct local architecture (32-bit or 64-bit).

3. Check that you are using the appropriate Java version (32-bit or 64-bit).
4. See the advice given above about editing or copying Oracle files.
5. Try reconnecting to the database.
6. Re-copy 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 passphrase is correct.
3. If OCS or Softcard key protection is used, ensure that 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-00600: internal error code

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

1. Ensure that you have added the `oracle` user to the `nfast` group. In some cases, you may need to log out and log back in as the `oracle` user for this to take effect.
2. If a FIPS 140 Level 3 Security World is in use, ensure that an OCS card is inserted in the HSM slot.

3.7. ORA-12162: TNS: Net service name is incorrectly specified

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

Chapter 4. Appendix

4.1. Security Worlds, key protection, and failure recovery

This section highlights key considerations when choosing Security World and key-protection options for use with the Entrust Security World. It focuses on recovery of Security World authorization when a system has temporarily failed (for example, after a power outage) and is then returned to operation. It does not cover other failure recovery functions. These considerations apply 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, see 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.
- FIPS authorization, if you are using a FIPS 140 Level 3 Security World.

Credential authorization can be granted using either a Softcard or an OCS card, with a passphrase. When using an OCS, a card must always be available in a valid HSM card reader so that reauthorization can be granted after a failure and automatic recovery can take place.

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

If you are using OCS cards through an RA secure channel, and the secure channel is lost, it must be re-established before recovery using the OCS cards can begin. There is no automatic mechanism to re-establish the secure channel: it must 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 must be set up to enable 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 cards 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. Use the same passphrase for all cards. (2) Leave one OCS card in the HSM slot. Recovery is automatic.	Use OCS for credential authorization: (1) Use 1/N quorum. Use the same passphrase for all cards. (2) Leave an OCS card in the slot of every HSM in the 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 cards 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 only, the quorum automatically defaults

to 1/N, and any passphrase is ignored.

Authorization acquired through a persistent operator card is not automatically reinstated after a temporary failure.

4.2. About the HSM credential

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

- **Module**: Encryption keys are protected by an nCipher Security World protecting key in the HSM.
- **Softcard**: Encryption keys are protected by a named Softcard (software-based) token key, a passphrase, and the nCipher Security World protecting key in the HSM. You can have many different Softcards, but each one is standalone.
- **OCS**: Encryption keys are protected by the presence of a named physical token (an OCS smartcard), an OCS token key, a passphrase, and the nCipher Security World protecting key in the HSM. OCS cards are usually part of a set of several OCS cards (a 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 the *User Guide* for your HSM for details. The module protection method can be used directly without any setup beyond the normal Entrust configuration. Setting up a Softcard or OCS includes creating and naming the tokens, each with a passphrase (see the *User Guide* for your HSM).

Within the SQL scripts 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 SQL uses the separator symbol | or : to divide the <credential-passphrase> and <credential-name>. The total Oracle SQL string for a credential is therefore:

- 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 (<credential-name>):

- 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 applies the following restrictions to credential passphrases:

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

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

The Oracle SQL interface imposes additional restrictions on the characters allowed in the string <credential-passphrase> + <separator> + <credential-name>:

- 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, and
- 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 each of the passphrase and the 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, any SQL script that uses the credential must get the `<credential-passphrase>` and `<credential-name>` exactly right. If there is a mistake, the entire credential string may be misinterpreted as a `<module-passphrase>`, and your encryption keys will then be placed under module protection rather than the Softcard or OCS protection you intended. For this reason, after creating encryption keys or rekeying, use the nCipher `rocs` utility immediately to check that 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 their purpose clear, 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 first change the passphrase for the token in the nCipher Security World, and then update the database to use the new passphrase. For module protection, you only need to change the passphrase as seen by the database.

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

- To change the passphrase of a Softcard, or to create a new Softcard, you need either authorization using ACS cards or an authorizing OCS card.
- To change the passphrase of an OCS card, or to create a new OCS card, you need 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 still change the passphrase. In this case, after the passphrase is changed, 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 between different Softcards, or between 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, run the following SQL:

   `CONNECT sysdba@CDB<n>$ROOT`

   ```
   -- If the database is not already open
   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, see 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, see 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 cards within the same (1/N) card set must be updated to share the same passphrase.
3. Bounce the database.
4. Run the following SQL:

   `CONNECT sysdba@CDB<n>$ROOT`

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

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

4.2.2. Change token with associated passphrase but keep the same protection method

This does not apply to module protection.

1. To change a token with a passphrase for the same protection method, run the following SQL:

   `CONNECT sysdba@CDB<n>$ROOT`

   ```
   -- If the database is not already open
   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 same passphrase.

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

     

     Entrust recommends that you 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 need your Security World ACS cards to authorize the transfer of keys between tokens. You can only transfer encryption keys between Softcards, or between OCS cards, but not between Softcards and OCS cards. If you are transferring keys to another OCS card set (1/N), all OCS cards within the target card set must share the same passphrase.
3. Bounce the database.
4. Run the following SQL:

   `CONNECT sysdba@CDB<n>$ROOT`

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

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

4.2.3. Change protection method

1. To change the protection method, run the following SQL:

   CONNECT sysdba@CDB<n>$ROOT

   ```
   -- If the database is not already open
   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 same passphrase.

2. Bounce the database.
3. Run the following SQL:

   CONNECT sysdba@CDB<n>$ROOT

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

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

4.3. Latency issues

It is beyond the scope of this guide to deal with specific solutions to latency issues; they are discussed here only 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 the Oracle symmetric keys, which are in turn used for tablespace or table column encryption. The Oracle symmetric keys are stored as part of the database itself, although they are protected by the wrapping key.

In the context of this guide, *encrypted data* refers to the symmetric keys that are stored as part of an Oracle database as well as the encrypted data itself. *Master encryption keys* refers to the wrapping keys stored by the nCipher Security World. Latency issues can occur when there is a mismatch between the encrypted data and the correct master encryption keys, due to a time lag in updating either of them. This should only be a problem where multiple clients are 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 to the other clients before they use them. Otherwise, synchronization problems may occur. Note that the client that initiates the changes should not experience synchronization problems.

Typically, these issues are more complex to resolve for a large and geographically distributed database system than for a small or localized system. It is your system

administrator's responsibility to ensure that encrypted data is synchronized with the appropriate master encryption keys at any given time. Furthermore, nCipher software cannot control whether encrypted data matches the correct master encryption keys in the Security World when database updates are delayed.

A time lag in updating master encryption keys in the Security World to match encrypted data may be due to either of the following:

- A 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.
- A lag in making an nCipher hardserver instance recognize the new master keys after they have been successfully distributed to the Security World.

4.3.1. Storage and distribution of updated master keys

4.3.1.1. Common storage of master encryption keys

Where possible, Entrust recommends configurations where the Security World data is held in common storage between clients that need to use the same master encryption keys.

If common storage of master encryption keys is used, there may be a short delay before newly created keys are successfully copied to the common store. After this, there may be a further short delay before a client can access the keys from the common store. The period during which a client cannot access the updated keys is likely to be very short, but may grow if the client is geographically distant from the common store and communication delays accumulate. With a common store, 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:

- Makes key updates implicit and simple, because there is only one store.
- Keeps time delays short, minimizing problems synchronizing keys with data.
- Requires the common store to be backed up frequently, because it is otherwise the only copy of the encryption keys.

4.3.1.2. Local storage of master encryption keys

If each client is using its own local copy of the Security World, then after a master key update 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 an explicit update mechanism to recognize when an update is required in the first place, and to then

trigger the key distribution process.

If this is 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, so triggering the update should not 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 in the common key storage case. This makes timely synchronization of the master keys with the data more challenging.

Entrust provides the utilities `rfs-setup` and `rfs-sync` (gang-client) that offer limited facilities to distribute keys between different clients, although you must use an RFS for intermediate key storage. These utilities were originally designed for manual operation, but can be incorporated into automated scripts customized for your particular configuration. Developing these scripts into a fully automated system to distribute your keys without synchronization problems is a task for your system development team. For more information about the nCipher `rfs-setup` and `rfs-sync` utilities, see 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 with distributed key update arrangements, contact Entrust Support.

Local key storage and distribution:

- Requires an explicit update mechanism that may be complex to automate.
- Makes it harder to keep distribution delays short, increasing the chance of problems synchronizing keys with data.
- Provides multiple copies of the Security World, so the loss of any one copy is less significant than it would be with common storage.

4.3.1.3. 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 are immediately available and usable on the client that performed the rekey. However, on other clients, once the new keys are available in the Security World folder, choose one of the following options to make the keys usable by the local hardserver instance (for both shared and local key storage):

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

```
CKNFAST_ASSUME_SINGLE_PROCESS=0
```

This ensures that 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 is scanned every time a key is required, even if no new keys have been added. If there are many keys, this can take significant time, and because it is repeated every time a key is needed, it can slow down overall operations. This option does not require downtime for the key update.

2. For each client that did not initiate a rekey, reconnect all applications and users that were using encryption keys on the database. A new connection forces a scan of the Security World, which picks up new keys. In this case, however, it is a single scan for that connection and is NOT repeated every time a key is required. If you have many keys, encrypted database operations will be temporarily affected only during the reconnection needed to update master keys. This option may imply temporary downtime while reconnections are made after a key update. However, if you routinely make new connections on your system per transaction, the impact should be barely noticeable.

4.3.1.4. Other considerations

Even if a client is unable 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 it is unable to obtain the correct master keys. It should be possible to program the database to roll back failed transactions and make several attempts to repeat the transaction, until some expiry point is reached.

If the delay in updating the master keys is short, 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, you should be able to do something similar in the application code that uses the database.

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

- The Oracle database is not affected.
- The Oracle database copes gracefully and subsequently recovers automatically, as described above, 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, it may be necessary to halt encryption transactions temporarily while a master key rotation is performed.

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

4.4. 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 servers 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 can 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 smartcards that must be used to create a Security World. Once 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 a Security World, load it 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 (a *quorum*) required to load the Security World onto an HSM at a later time. The number of cards in the set should exceed the quorum, so that spares are available in case of card failure or loss. An encrypted copy of the created Security World is stored in a folder on the host servers.

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, with their associated passphrases, 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 can 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 named software token key (a single Softcard), a passphrase, and the Security World protecting key.
- **Operator Card Set (OCS) protection**: Oracle master encryption keys are protected by the presence of a set of named physical tokens (smartcards), an OCS token key, and the Security World protecting key. An OCS smartcard set is similar to the ACS card set in that it stipulates a quorum of cards required to authorize use of its protection. The number of cards in the set should exceed the number of HSMs that may share 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 on setting up these protection methods, see the *User Guide* for your HSM.

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

In this *Integration Guide*, the word *credential* is used for a passphrase, or for the combination of a passphrase and a named token (OCS or Softcard). Before an Oracle database can use 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: without access to the nCipher library file, the Oracle database and nShield HSM (or nCipher software) cannot communicate. Once communication is established between the Oracle database and the nShield HSM, the Oracle database can gain access to the HSM through a credential incorporated into an SQL script. When it is set up with a credential, the Oracle database can create and assign encryption keys to that credential if no encryption keys yet exist, and can encrypt or decrypt data using the encryption keys protected by that credential.

A protection method or credential is uniquely associated with the Security World in which it was created, and cannot be used with any other Security World. It should also be uniquely associated with the encrypted databases it protects. 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 only for the Oracle database access

mechanisms. The Oracle module protection passphrase does not have a reference or counterpart in the nShield HSM. This means that a user who can access keys directly in the HSM can access module-protected keys for any database without needing the Oracle passphrase. This does not apply to Softcard or OCS protection.

Use of HSM credentials and the associated SQL scripts that open up access to 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, which is automatically decrypted when it leaves the database. Approved database users do not need the HSM credentials or 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. Encryption facilities can also be migrated from an HSM back to an Oracle software keystore. During migration, fresh master keys are created in the HSM or software keystore, and the subsidiary keys being protected are re-encrypted with the new master keys. 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 load balancing 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 has 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 certified to FIPS 140 Level 3, meaning that they are tamper-evident and tamper-resistant. nShield Connect units 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 it is stored on disk, but once the data is retrieved into 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, so there should be minimal security risk in transmitting it over open networks. Similarly, encrypted database

contents should be at minimal risk if transmitted over open networks.

Chapter 5. Additional resources and related products

5.1. Entrust products

5.2. nShield product documentation