Oracle® Database

Transparent Data Encryption Guide

23ai
G12881-03
April 2025

Oracle Database Transparent Data Encryption Guide, 23ai

G12881-03

Contents

3 Configuring United Mode

7 Managing the Keystore and the Master Encryption Key

8 Administering United Mode

9 Administering Isolated Mode

10 General Considerations of Using Transparent Data Encryption

11 Using Transparent Data Encryption with Other Oracle Features

Preface

Welcome to the *Oracle Database Transparent Data Encryption Guide*. This guide describes how to implement, configure, and administer Transparent Data Encryption (TDE).

Audience

Oracle Database Transparent Data Encryption Guide is intended for users and systems professionals involved with the implementation, configuration, and administration of Transparent Data Encryption including:

- Implementation consultants
- System administrators
- Security administrators
- Database administrators (DBAs)

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customer access to and use of Oracle support services will be pursuant to the terms and conditions specified in their Oracle order for the applicable services.

Diversity and Inclusion

Oracle is fully committed to diversity and inclusion. Oracle respects and values having a diverse workforce that increases thought leadership and innovation. As part of our initiative to build a more inclusive culture that positively impacts our employees, customers, and partners, we are working to remove insensitive terms from our products and documentation. We are also mindful of the necessity to maintain compatibility with our customers' existing technologies and the need to ensure continuity of service as Oracle's offerings and industry standards evolve. Because of these technical constraints, our effort to remove insensitive terms is ongoing and will take time and external cooperation.

Related Documents

Before you configure Oracle Advanced Security features, you should be familiar with the following guides:

- *Oracle Database Administrator’s Guide*
- *Oracle Database Security Guide*

- *Oracle Database SQL Language Reference*
- *Oracle Database Reference*
- *Oracle Database PL/SQL Packages and Types Reference*
- *Oracle Multitenant Administrator's Guide*

Many books in the documentation set use the sample schemas of the default database. Refer to *Oracle Database Sample Schemas* for information about how these schemas were created and how you can use them.

Oracle Technical Services

To download the product data sheet, frequently asked questions, links to the latest product documentation, product download, and other collateral, visit Oracle Technical Resources (formerly Oracle Technology Network). You must register online before using Oracle Technical Services. Registration is free and can be done at

`https://www.oracle.com/technical-resources/`

My Oracle Support

You can find information about security patches, certifications, and the support knowledge base by visiting My Oracle Support (formerly Oracle*MetaLink*) at

`https://support.oracle.com`

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

Changes in This Release for Oracle Database Transparent Data Encryption Guide

This preface contains:

Changes in Oracle Database Transparent Data Encryption 23ai

Oracle Database Transparent Data Encryption Guide for Oracle Database 23ai has new security features.

Changes for Encryption Algorithms and Modes

Starting with Oracle Database 23ai, the default encryption algorithms and the encryption modes have changed.

Encryption algorithm changes:

- Encryption algorithm changes:
  - The default encryption algorithm for both TDE column encryption and TDE tablespace encryption is now AES256. The previous default for TDE column encryption was `AES192`. For TDE tablespace encryption, the default was `AES128`.
  - The **decryption libraries** for the `GOST` and `SEED` algorithms are deprecated. New keys cannot use these algorithms. The **encryption libraries** for both of these libraries are desupported. The GOST decryption libraries are desupported on HP Itanium platforms.
- The column encryption mode is now Galois/Counter mode (GCM) instead of cipher block chaining (CBC), and in tablespace encryption, you can choose between the new "tweakable block ciphertext stealing (XTS)" operating mode or cipher feedback (CFB). XTS is the default.
- The Oracle Recovery Manager (Oracle RMAN) integrity check for column encryption keys now uses `SHA512` instead of `SHA1`.
- The keys for Oracle RMAN and column keys are now derived from `SHA512`/AES for key generation. In previous releases, they used `SHA-1`/3DES as a pseudo-random function.

These enhancements enable your Oracle Database environment to use the latest, most secure algorithms and encryption modes.

Related Topics

- Supported Encryption and Integrity Algorithms
  Oracle supports the AES, ARIA and DES algorithms.

AES-XTS Encryption Mode Support for TDE Tablespace Encryption

Starting with Oracle Database 23ai, Transparent Database Encryption (TDE) tablespace encryption supports Advanced Encryption Standard (AES) XTS (XEX-based mode with ciphertext stealing mode) in the `CREATE TABLESPACE` and `ALTER TABLESPACE` statements.

AES-XTS provides improved security and better performance, especially on platforms where TDE can take advantage of parallel processing and specialized instructions built into processor hardware.

Related Topics

- *Oracle Database SQL Language Reference*

Oracle Data Guard Redo Decryption for Hybrid Disaster Recovery Configurations

Available with Oracle Database 23ai, Oracle Data Guard enables you to decrypt redo operations in hybrid cloud disaster recovery configurations where the Cloud database is encrypted with TDE and the on-premises database is not.

To enable this feature, Oracle Database introduces the `TABLESPACE_ENCRYPTION` initialization parameter, which enables you to control the automatic encryption of tablespaces in both the primary and standby databases, for on-premises and Oracle Cloud Infrastructure (OCI) environments. For example, an on-premises database can be unencrypted and an OCI database can be encrypted.

Hybrid disaster recovery is often considered a quick-stepping stone to cloud adoption. By enabling the ability to quickly configure disaster recovery even in cases where on-premises databases might not already be encrypted with TDE, the steps required to configure hybrid disaster recovery environments are reduced while still ensuring that redo data is still encrypted during the transportation process.

Related Topics

- Encryption of Tablespaces in an Oracle Data Guard Environment
  You can control tablespace encryption in the primary and standby databases in an Oracle Data Guard environment.
- Hybrid Oracle Data Guard without Transparent Data Encryption (TDE) License

Updates to Oracle Database Transparent Data Encryption 23ai

Oracle Database Transparent Data Encryption Guide for Oracle Database 23ai as the following update.

New Parameter to Control the TDE Rekey Operations for Oracle Data Guard

You now can use the `DB_RECOVERY_AUTO_REKEY` initialization parameter for Oracle Data Guard environments..

`DB_RECOVERY_AUTO_REKEY` controls whether an Oracle Data Guard standby database recovery operation automatically performs the corresponding tablespace rekey when it encounters a redo that says the primary database has performed a tablespace rekey operation.

This feature is useful for standby deployments with large tablespaces whose users must perform an online TDE conversion.

Related Topics

- Uncoupling the Standby Database from the Primary Database Online Encryption Process
  You can use the `DB_RECOVERY_AUTO_REKEY` initialization parameter to control how Transparent Data Encryption (TDE) rekey operations are performed in an Oracle Data Guard environment.

1

Quick-start Setup Guide for Wallet Based Transparent Data Encryption

1. Create the directories that will hold the TDE wallet (a PKCS#12 container that is encrypted with a key that is derived from the TDE wallet password).
   The last two commands change the ownership of the directories to `oracle:oinstall` and reduce the file privileges to the minimum:

   The `ORACLE_SID` in this example is `finance`:

   ```
   mkdir -pv /etc/ORACLE/KEYSTORES/finance/tde_seps
   chown -Rv oracle:oinstall /etc/ORACLE
   chmod -Rv 700 /etc/ORACLE
   ```

2. Set static system parameter `WALLET_ROOT` to the directory that you just created:

   ```
   SYS> alter system set WALLET_ROOT = '/etc/ORACLE/KEYSTORES/$ORACLE_SID'
   scope = spfile;
   ```

3. Set the static `TABLESPACE_ENCRYPTION` parameter to `AUTO_ENABLE`, so that all new tablespaces are encrypted, even if the `encryption` key-words are not part of the `create tablespace` commands:

   ```
   SYS> alter system set TABLESPACE_ENCRYPTION = AUTO_ENABLE scope = spfile;
   ```

4. Restart the database to activate those two parameters.
5. The next parameter configures the database to use a TDE wallet for file-based TDE setup:

   ```
   SYS> alter system set TDE_CONFIGURATION = "KEYSTORE_CONFIGURATION=FILE"
   scope = both;
   ```

6. Create a new password-protected and local auto-open TDE wallet; the local auto-open wallet enables automatic database restarts without DBA intervention to open the password-protected TDE wallet:

   > **Note:**
   >
   > Do not lose your wallet password. You should record the password, protecting it according to your organization's standards for sensitive IT secrets

(This command also creates the `<WALLET_ROOT>/tde directory`)

```
SYSKM> administer key management CREATE KEYSTORE identified by <wallet-
pwd>;


SYSKM> administer key management CREATE LOCAL AUTO_LOGIN KEYSTORE from
keystore identified by <wallet-pwd>;
```

7. Add the TDE wallet password as a secret into another (local) auto-open wallet in `<WALLET_ROOT>/tde_seps`. This allows you to hide the TDE wallet password from the SQL*Plus command line and replace it with `EXTERNAL STORE`:

```
SYSKM> administer key management ADD SECRET '<wallet-pwd>' for client
'TDE_WALLET' to LOCAL auto_login keystore '/etc/ORACLE/KEYSTORES/finance/
tde_seps';
```

8. In the root container database, set the first TDE master key:

```
SYSKM> administer key management SET KEY force keystore identified by
EXTERNAL STORE with backup container = current;
```

9. Encrypt the tables in the root CDB:

```
SYS:CDB$ROOT>
alter tablespace USERS encryption ONLINE encrypt;
alter tablespace SYSTEM encryption ONLINE encrypt;
alter tablespace SYSAUX encryption ONLINE encrypt;
```

10. Define either a united or isolated keystore for the PDB:
    - **United Keystore** In the PDB, set the first TDE master key:

      ```
      SYSKM:FINPDB23AI> administer key management SET KEY force keystore
      identified by EXTERNAL STORE with backup;
      ```

    - **Isolated Keystore**
      a. From the PDB, create an isolated keystore with its own keystore password:

         ```
         SYSKM:FINPDB23AI> administer key management CREATE KEYSTORE
         identified by <PDB-wallet-pwd>;
         ```

         The previous command does three things:
         i. It sets `TDE_CONFIGURATION` to `FILE` for the isolated PDB
         ii. It creates the `<PDB_GUID>/tde` directories under `<WALLET_ROOT>`
         iii. It creates an individual wallet for the PDB, with its own TDE wallet password (that is potentially unknown to the DBA of the root container)

      b. Create a (local) auto-open wallet for the isolated PDB:

         ```
         SYSKM:FINPDB23AI> administer key management CREATE LOCAL AUTO_LOGIN
         KEYSTORE from keystore identified by <PDB-wallet-pwd>;
         ```

c. Create the directory `<WALLET_ROOT>/<PDB_GUID>/tde_seps` by executing the output of the following command:

```
SYS:FINPDB23AI> select ' host mkdir -pvm700 '''||v.value||'/'||
guid||'/tde_seps'';' from v$pdbs, v$parameter v where v.name like
'%root%';
```

d. Add the TDE wallet password as a secret into the wallet in `<WALLET_ROOT>/<PDB_GUID>/tde_seps` by executing the output of the following command. This allows you to hide the TDE wallet password of the isolated PDB from the SQL*Plus command line and replace it with `EXTERNAL STORE`:

```
SYS:FINPDB23AI> select ' administer key management ADD SECRET
''<PDB-wallet-pwd>'' for client ''TDE_WALLET'' to LOCAL auto_login
keystore '''||v.value||'/'||guid||'/tde_seps/'';' from v$pdbs,
v$parameter v where v.name like '%root%';
```

11. Encrypt the tablespaces in the PDB:

```
SYS:FINPDB23AI> alter tablespace USERS encryption ONLINE encrypt;
SYS:FINPDB23AI> alter tablespace SYSTEM encryption ONLINE encrypt;
SYS:FINPDB23AI> alter tablespace SYSAUX encryption ONLINE encrypt;
```

12. Confirm:

```
SYS> select distinct c.name as PDB_NAME, t.name as TBS_NAME,
nvl(e.encryptionalg, '----') as
ENC_ALG, nvl(e.ciphermode, '---') as "MODE", nvl(e.status, '----') as
ENC_STATUS from
v$containers c, v$tablespace t, v$encrypted_tablespaces e where (c.con_id !
= 2) and e.ts#(+) =
t.ts# and c.con_id(+)=t.con_id order by 1, 3 desc, 2;


PDB_NAME        TBS_NAME             ENC_ALG   MODE  ENC_STATUS
--------------- -------------------- --------- ----- -------
CDB$ROOT        SYSAUX               AES256    XTS   NORMAL
CDB$ROOT        SYSTEM               AES256    XTS   NORMAL
CDB$ROOT        USERS                AES256    XTS   NORMAL
CDB$ROOT        TEMP                 ----      ---   ----
CDB$ROOT        UNDOTBS1             ----      ---   ----
FINPDB23AI      SYSAUX               AES256    XTS   NORMAL
FINPDB23AI      SYSTEM               AES256    XTS   NORMAL
FINPDB23AI      USERS                AES256    XTS   NORMAL
FINPDB23AI      TEMP                 ----      ---   ----
FINPDB23AI      UNDOTBS1             ----      ---   ----
```

2

Introduction to Transparent Data Encryption

Transparent data encryption enables you to encrypt database data files or selected columns of data. This helps you protect sensitive data contained in your database, such as credit card numbers or Social Security numbers.

2.1 What Is Transparent Data Encryption?

Transparent Data Encryption (TDE) enables you to encrypt sensitive data that you store in tables and tablespaces. It also enables you to encrypt database backups.

After the data is encrypted, this data is transparently decrypted for authorized users or applications when they access this data. TDE helps protect data stored on media (also called data at rest) in the event that the storage media or data file is stolen.

Oracle Database uses authentication, authorization, and auditing mechanisms to secure data in the database, but not in the operating system data files where data is stored. To protect these data files, Oracle Database provides Transparent Data Encryption (TDE). TDE encrypts sensitive data stored in data files. To prevent unauthorized decryption, TDE stores the encryption keys in a security module that is external to the database. This security module can be referred to as follows:

- **TDE wallets** are wallets used for TDE. They cannot contain other security artifacts such as certificates. In previous releases, they were called software keystores or just wallets.
- **External keystores** refer to Oracle Key Vault or Oracle Cloud Infrastructure (OCI) Key Management Service (KMS).
- **Keystores** is a generic term for both TDE wallets and external keystores.

You can configure Oracle Key Vault as part of the TDE implementation. This enables you to centrally manage keystores in your enterprise. For example, you can upload a TDE wallet to Oracle Key Vault, migrate the database to use Oracle Key Vault as the default keystore, and then share the contents of this keystore with other primary and standby Oracle Real Application Clusters (Oracle RAC) nodes of that database to streamline daily database administrative operations with encrypted databases.

> **Note:**
>
> Tranparent Data Encryption (TDE) public key infrastructure (PKI) keys are desupported with Oracle Database 23ai.

Related Topics

- *Oracle Key Vault Administrator's Guide*

2.2 How Configuring Transparent Data Encryption Works

To configure Transparent Data Encryption, you must perform a one-time setup before you create keystores and encrypt data.

Before you can begin to encrypt data, you must perform a one-time configuration using the static `WALLET_ROOT` parameter and the dynamic `TDE_CONFIGURATION` parameter to designate the location and type of keystores that you plan to use. See *Oracle Database Reference* for details about the WALLET_ROOT and TDE-CONFIGURATION parameters.

The `WALLET_ROOT` parameter specifies the keystore directory location. Before you set `WALLET_ROOT`, ensure that you have an existing directory that you can use to store keystores (for example, `WALLET_ROOT` can be set to the existing directory `/etc/ORACLE/KEYSTORES/${ORACLE_SID}`).

The `TDE_CONFIGURATION` parameter specifies the type of keystore (TDE wallet or Oracle Key Vault). After you set the type of keystore using `TDE_CONFIGURATION`, when you create the keystore, Oracle Database creates a directory within the `WALLET_ROOT` location for the keystore type. For example, if you set `TDE_CONFIGURATION` to `FILE`, then Oracle Database creates a TDE wallet in `WALLET_ROOT/tde`. To use Oracle Key Vault, install the Oracle Key Vault client software into `WALLET_ROOT/okv` and set `TDE_CONFIGURATION` to `OKV`. An auto-open connection helps with automatically restarting databases (for example, primary and standby databases that switch roles (planned or unplanned)). Oracle Real Application Clusters (Oracle RAC) databases should also have an auto-open connection into Oracle Key Vault to not interfere with an automated Oracle RAC instance restart. To establish an auto-open Oracle Key Vault configuration, add the Oracle Key Vault password as a secret into a (local) auto-open wallet in *`WALLET_ROOT`*`/tde`. The Oracle Key Vault password is the password that you created when you installed the Oracle Key Vault endpoint software. If you want to migrate from one keystore type to another, then you must first set `TDE_CONFIGURATION` parameter to the keystore type that you want to use, and then use the `ADMINISTER KEY MANAGEMENT` statement to perform the migration. For example, you can migrate from a TDE wallet to Oracle Key Vault.

The `KEYSTORE_MODE` column of the `V$ENCRYPTION_WALLET` dynamic view shows whether united mode or isolated mode has been configured. For example, from a PDB:

```
SELECT KEYSTORE_MODE FROM V@ENCRYPTION_WALLET;

KEYSTORE
--------
UNITED
```

From the root, you can find how all containers are configured:

```
SELECT KEYSTORE_MODE, CON_ID FROM V@ENCRYPTION_WALLET;

KEYSTORE  CON_ID
--------  ------
NONE           1
UNITED         2
UNITED         3
```

> **Note:**
>
> The configuration of TDE using the `sqlnet.ora` file is desupported. Upgrades of Oracle Database 19c and 21c databases to release 23ai will fail if the source databases do not have the `WALLET_ROOT` and `TDE_CONFIGURATION` parameters set.

Related Topics

- Transparent Data Encryption Keystore Search Order
  The search order for the TDE keystore depends on how you have set either the instance initialization parameters, the `sqlnet.ora` parameters, or the environment variables.

2.3 Benefits of Using Transparent Data Encryption

Transparent Data Encryption (TDE) ensures that sensitive data is encrypted, helps address compliance requirements, and provides functionality that streamlines encryption operations.

Benefits are as follows:

- As a security administrator, you can be sure that sensitive data is encrypted and therefore safe in the event that the storage media or data file is stolen, or when an intruder tries to access the data files from the operating system, bypassing the access controls of the database.
- Using TDE helps you address security-related regulatory compliance issues.
- You do not need to create auxiliary tables, triggers, or views to decrypt data for the authorized user or application. Data from tables is transparently decrypted for the database user and application. An application that processes sensitive data can use TDE to provide strong data encryption with little or no change to the application.
- Data is transparently decrypted for database users and applications that access this data. Database users and applications do not need to be aware that the data they are accessing is stored in encrypted form.
- Using online or offline encryption of existing un-encrypted tablespaces enables you to implement Transparent Data Encryption with little or no downtime.
- You do not need to modify your applications to handle the encrypted data. The database manages the data encryption and decryption.
- Oracle Database automates TDE master encryption key and keystore management operations. The user or application does not need to manage TDE master encryption keys.

2.4 Who Can Configure Transparent Data Encryption?

To configure Transparent Data Encryption (TDE), you must be granted either the `SYSKM` administrative privilege or the `ADMINISTER KEY MANAGEMENT` system privilege.

A user can authenticate as a security administrator in two ways, which are quite different, and both authentication approaches allow the security administrator to issue the `ADMINISTER KEY MANAGEMENT` statement. To check if a user is allowed to perform a TDE operation, Oracle Database checks whether the user has the `ADMINISTER KEY MANAGEMENT` privilege or was authenticated using the `SYSKM` administrative privilege. Any `ADMINISTER KEY MANAGEMENT` statement will be allowed for the following users:

- A user who is granted the SYSKM administrative privilege (this grant is recorded in the password file) and issues the ADMINISTER KEY MANAGEMENT statement from a session that connected to the database using the AS SYSKM clause. For example:

  ```
  sqlplus c##sec_admin AS SYSKM
  Enter password: password
  ```

  The reason for this is that ADMINISTER KEY MANAGEMENT statements need to be supported when the Oracle data dictionary is not available. The only way to do the necessary privilege check in that situation is to check for the SYSKM administrative privilege in the password file. Note that the system needs to be configured with a password file in order to be able to grant the SYSKM administrative privilege to a user. If the user needs to perform ADMINISTER KEY MANAGEMENT operations such as opening the TDE keystore when the Oracle database is in the MOUNTED state, then the user must be granted the SYSKM administrative privilege.
- A user who has been granted the ADMINISTER KEY MANAGEMENT system privilege, and logs in to the database *without* the AS SYSKM clause. For example:

  ```
  sqlplus c##sec_officer
  Enter password: password
  ```

To use TDE, you do not need the SYSKM or ADMINISTER KEY MANAGEMENT privileges. You must have the following privileges to encrypt table columns and tablespaces:

- CREATE TABLE
- ALTER TABLE
- CREATE TABLESPACE
- ALTER TABLESPACE (for online and offline tablespace encryption)
- ALTER DATABASE (for offline tablespace encryption)

2.5 Types and Components of Transparent Data Encryption

Transparent Data Encryption can be applied to individual columns or entire tablespaces.

2.5.1 About Transparent Data Encryption Types and Components

You can encrypt sensitive data at the column level or the tablespace level.

At the column level, you can encrypt sensitive data in application table columns. TDE tablespace encryption enables you to encrypt all of the data that is stored in a tablespace.

Both TDE column encryption and TDE tablespace encryption use a two-tiered key-based architecture. Unauthorized users, such as intruders who are attempting security attacks, cannot read the data from storage and back up media unless they have the TDE master encryption key to decrypt it.

2.5.2 How Transparent Data Encryption Tablespace Encryption Works

Transparent Data Encryption (TDE) tablespace encryption enables you to encrypt an entire tablespace.

All of the objects that are created in the encrypted tablespace are automatically encrypted. TDE tablespace encryption is useful if your tables contain sensitive data in multiple columns, or

if you want to protect the entire table and not just individual columns. You do not need to perform a granular analysis of each table column to determine the columns that need encryption.

In addition, TDE tablespace encryption takes advantage of bulk encryption and caching to provide enhanced performance. The actual performance impact on applications can vary.

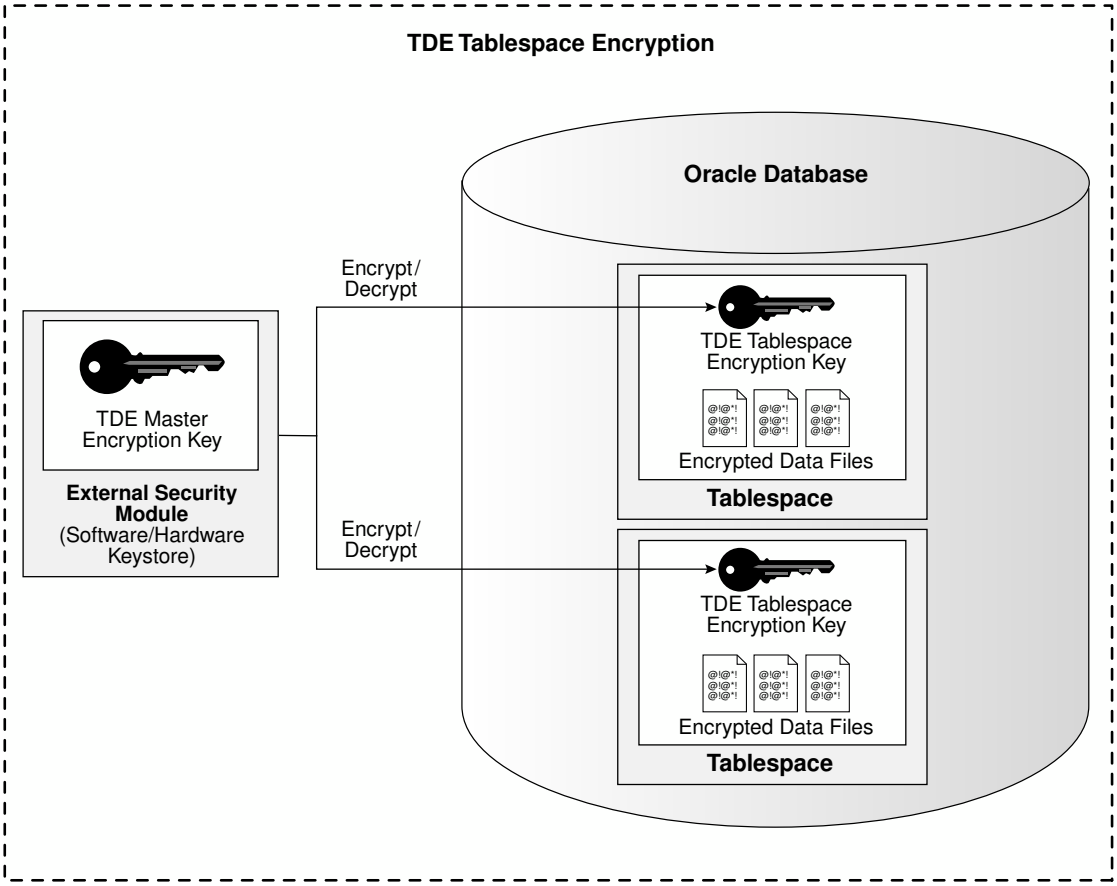
TDE tablespace encryption encrypts all of the data stored in an encrypted tablespace including its redo data. TDE tablespace encryption does not encrypt data that is stored outside of the tablespace. For example, `BFILE` data is not encrypted because it is stored outside the database. If you create a table with a `BFILE` column in an encrypted tablespace, then this particular column will not be encrypted.

All of the data in an encrypted tablespace is stored in encrypted format on the disk. Data is transparently decrypted for an authorized user having the necessary privileges to view or modify the data. A database user or application does not need to know if the data in a particular table is encrypted on the disk. In the event that the data files on a disk or backup media is stolen, the data remains protected.

TDE tablespace encryption uses the two-tiered, key-based architecture to transparently encrypt (and decrypt) tablespaces. The TDE master encryption key is stored in a security module (Oracle wallet, Oracle Key Vault, or Oracle Cloud Infrastructure (OCI) key management service (KMS)). This TDE master encryption key is used to encrypt the TDE tablespace encryption key, which in turn is used to encrypt and decrypt data in the tablespace.

Figure 2-1 shows an overview of the TDE tablespace encryption process.

Figure 2-1 TDE Tablespace Encryption



> **Note:**
>
> The encrypted data is protected during operations such as `JOIN` and `SORT`. This means that the data is safe when it is moved to temporary tablespaces. Data in undo and redo logs is also protected.

TDE tablespace encryption also allows index range scans on data in encrypted tablespaces. It does not interfere with Exadata Hybrid Columnar Compression (EHCC), Oracle Advanced Compression, or Oracle Recovery Manager (Oracle RMAN) compression. This is not possible with TDE column encryption.

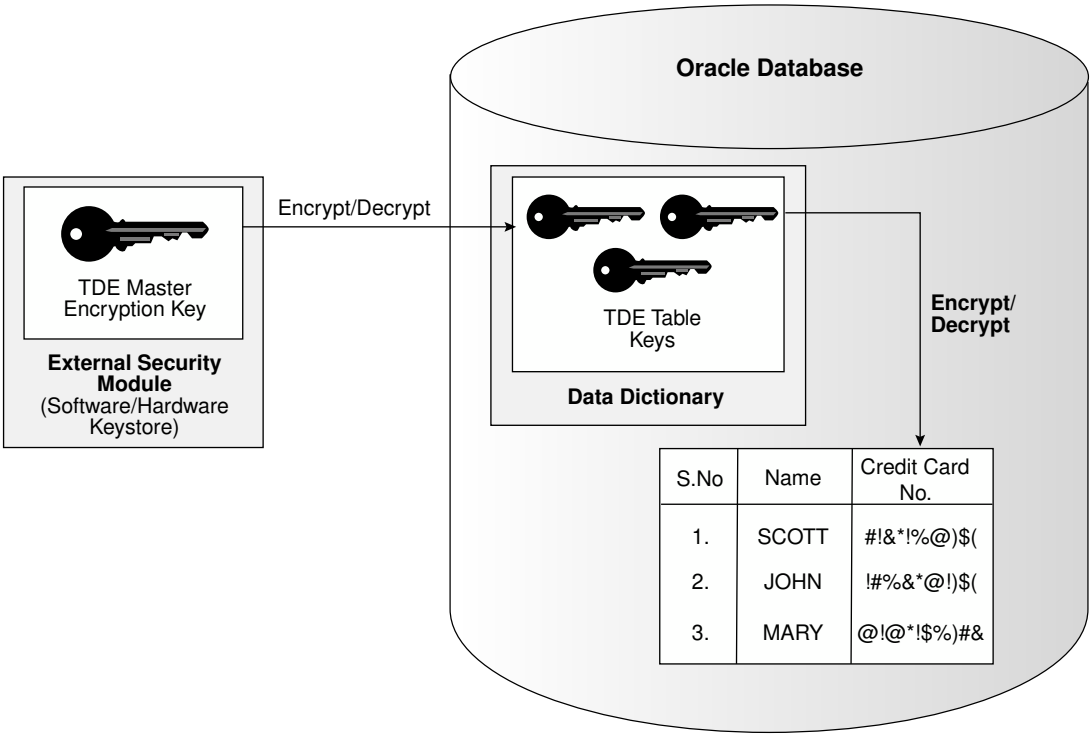
2.5.3 How Transparent Data Encryption Column Encryption Works

Transparent Data Encryption (TDE) column encryption protects confidential data, such as credit card and Social Security numbers, that is stored in table columns.

TDE column encryption uses the two-tiered key-based architecture to transparently encrypt and decrypt sensitive table columns. The TDE master encryption key is stored in an external keystore, which can be an Oracle wallet or in Oracle Key Vault. The Oracle wallet is a PKCS#12 container for certificates and encryption keys. It is encrypted with an AES256 key that is derived from the TDE wallet password. This TDE master encryption key encrypts and decrypts the TDE table key, which in turn encrypts and decrypts data in the table column.

Figure 2-2 shows an overview of the TDE column encryption process.

Figure 2-2 TDE Column Encryption Overview



As shown in Figure 2-2, the TDE master encryption key is stored in an external security module that is outside of the database and accessible only to a user who was granted the appropriate privileges. For this external security module, Oracle Database uses an Oracle TDE wallet (TDE wallet, in previous releases) or Oracle Key Vault. Storing the TDE master encryption key in this way prevents its unauthorized use.

Using an external security module separates ordinary program functions from encryption operations, making it possible to assign separate, distinct duties to database administrators and security administrators. Security is enhanced because the keystore password can be unknown to the database administrator, requiring the security administrator to provide the password.

When a table contains encrypted columns, TDE uses a single TDE table key regardless of the number of encrypted columns. Each TDE table key is individually encrypted with the TDE master encryption key.

2.5.4 How the Keystore for the Storage of TDE Master Encryption Keys Works

To control the encryption, you use a keystore and a TDE master encryption key.

2.5.4.1 About the Keystore Storage of TDE Master Encryption Keys

Oracle Database provides a key management framework for Transparent Data Encryption (TDE) that stores and manages keys and credentials.

The key management framework includes the keystore to securely store the TDE master encryption keys and the management framework to securely and efficiently manage keystore and key operations for various database components.

The Oracle keystore stores a history of retired TDE master encryption keys, which enables you to rotate the TDE master encryption key, and still be able to decrypt data (for example, for incoming Oracle Recovery Manager (Oracle RMAN) backups) that was encrypted under an earlier TDE master encryption key.

2.5.4.2 Benefits of the Keystore Storage Framework

The key management framework provides several benefits for Transparent Data Encryption.

- Enables separation of duty between the database administrator and the security administrator who manages the keys. You can grant the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege to users who are responsible for managing the keystore and key operations.
- Facilitates compliance, because it helps you to track encryption keys and implement requirements such as keystore password rotation and TDE master encryption key re-key operations. Both wallet password rotations and TDE master key re-key operation do not require database or application downtime.
- Facilitates and helps enforce keystore backup requirements. A backup is a copy of the password-protected TDE wallet that is created for all of the critical keystore operations.

  The mandatory `WITH BACKUP` clause of the `ADMINISTER KEY MANAGEMENT` statement creates a backup of the password-protected wallet before the changes are applied to the original password-protected wallet.
- Enables the keystore to be stored on an Oracle Automatic Storage Management (Oracle ASM) file system. This is particularly useful for Oracle Real Application Clusters (Oracle

RAC) environments where database instances share a unified file system view. In Oracle RAC, you must store the Oracle wallet in a shared location (Oracle ASM or Oracle Advanced Cluster File System (ACFS)), to which all Oracle RAC instances that belong to one database, have access to. Individual TDE wallets for each Oracle RAC instances are not supported.

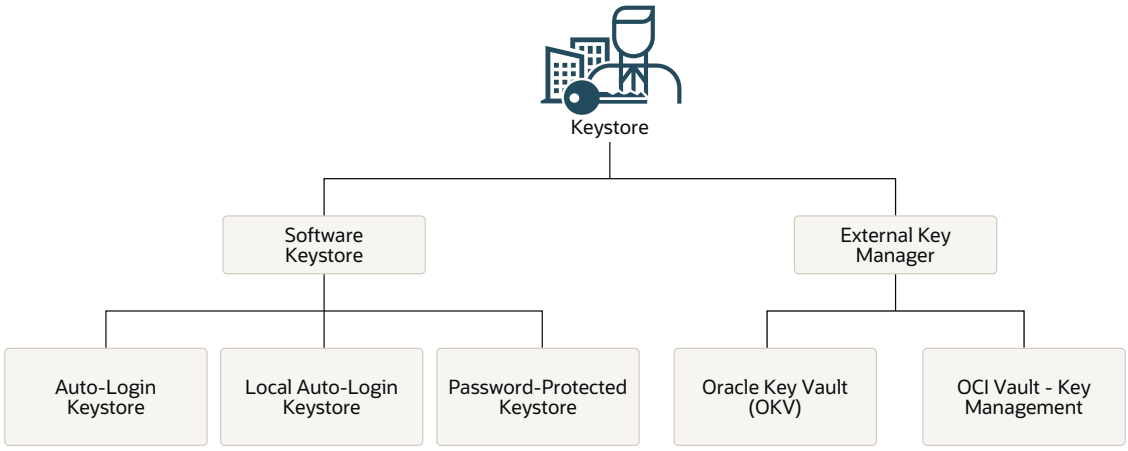
- Enables reverse migration from an external keystore to a file system-based TDE wallet. This option is useful if you must migrate back to a TDE wallet.

2.5.4.3 Types of Keystores

Oracle Database supports TDE wallets, Oracle Key Vault, and Oracle Cloud Infrastructure (OCI) key management systems (KMS).

Figure 2-3 illustrates the types of keystores that Oracle Database supports.

Figure 2-3 Oracle Database Supported Keystores



These keystores are as follows:

- **Auto-login TDE wallets:** Auto-login TDE wallets are protected by a system-generated password, and do not need to be explicitly opened by a security administrator. Auto-login TDE wallets are automatically opened when accessed at database startup. Auto-login TDE wallets can be used across different systems. If your environment does not require the extra security provided by a keystore that must be explicitly opened for use, then you can use an auto-login TDE wallet. Auto-login TDE wallets are ideal for unattended scenarios (for example, Oracle Data Guard standby databases).
- **Local auto-login TDE wallets:** Local auto-login TDE wallets are auto-login TDE wallets that are local to the computer on which they are created. Local auto-login TDE wallets cannot be opened on any computer other than the one on which they are created. This type of keystore is typically used for scenarios where additional security is required (that is, to limit the use of the auto-login for that computer) while supporting an unattended operation. You cannot use local auto-open wallets in Oracle RAC-enabled databases, because only shared wallets (in ACFS or ASM) are supported.
- **Password-protected TDE wallets:** Password-protected TDE wallets are protected by using a password that you create. You must open this type of wallet before the keys can be retrieved or used, and use a password to open this type of keystore..

TDE wallets can be stored in Oracle Automatic Storage Management (Oracle ASM), Oracle Advanced Cluster File System (Oracle ACFS), or regular file systems.

Under External Keystore Manager are the following categories:

- **Oracle Key Vault (OKV):** Oracle Key Vault is a software appliance that provides continuous key availability and scalable key management through clustering with up to 16 Oracle Key Vault nodes, potentially deployed across geographically distributed data centers. It is purpose-built for Oracle Database and its many deployment models (Oracle RAC, Oracle Data Guard, Exadata, multitenant environments). In addition, Oracle Key Vault provides online key management for Oracle GoldenGate encrypted trail files and encrypted ACFS. It is also certified for ExaDB-C@C and Autonomous Database (dedicated) (ADB-C@C). Oracle Key Vault is distributed as a full-stack software appliance for installation on dedicated hardware. It is also available in the OCI Marketplace and can be deployed in your OCI tenancy quickly and easily. See the video Deploying Oracle Key Vault in OCI.
- **OCI Vault - Key Management:** The Oracle Cloud Infrastructure (OCI) Key Management Service (KMS) is a cloud-based service that provides centralized management and control of encryption keys for data stored in OCI. It enables integration of encryption with other OCI services such as storage, database, and Fusion Applications for protecting data stored in these services.

Related Topics

- *Oracle Key Vault Administrator's Guide*

2.5.5 Supported Encryption and Integrity Algorithms

Oracle supports the AES, ARIA and DES algorithms.

> **Note:**
>
> Starting with Oracle Database 23ai, the following updates are in place:
>
> - The Transparent Data Encryption (TDE) decryption libraries for the GOST and SEED algorithms are deprecated.
> - The Transparent Data Encryption (TDE) encryption libraries for the GOST and SEED algorithms are desupported and removed.
> - The GOST decryption libraries are desupported on HP Itanium platforms.

The default encryption algorithm for TDE column encryption and TDE tablespace encryption is `AES256`.

For TDE column encryption, salt is added by default to plaintext before encryption unless specified otherwise. You cannot add salt to indexed columns that you want to encrypt. For indexed columns, choose the `NO SALT` parameter for the SQL `ENCRYPT` clause.

You can change encryption algorithms and encryption keys on existing encrypted columns by setting a different algorithm with the SQL `ENCRYPT` clause.

Table 2-1 lists the supported encryption algorithms.

Table 2-1 Supported Encryption Algorithms for Transparent Data Encryption

Algorithm	Key Size	Parameter Name
Advanced Encryption Standard (AES)	• 128 bit • 192 bits • 256 bits (default for TDE column encryption and TDE tablespace encryption)	• AES128 • AES192 • AES256
ARIA	• 128 bits • 192 bits • 256 bits	• ARIA128 • ARIA192 • ARIA256
Triple Encryption Standard (DES)	168 bits	3DES168

For integrity protection of TDE column encryption, the `SHA-1` hashing algorithm is used.

`3DES168` is maintained for backward compatibility and should not be used in new deployments; data encrypted with `3DES168`, as well as with `AES128` and `AES192`, should be re-keyed to `AES256` for its stronger post-quantum crypto resilience. Also, `3DES168` does not benefit from hardware crypto acceleration available in modern hardware.

Related Topics

- Setting the Tablespace Encryption Default Algorithm
  The `TABLESPACE_ENCRYPTION_DEFAULT_ALGORITHM` applies to specific encryption scenarios.

2.6 Transparent Data Encryption in a Multitenant Environment

In a multitenant environment, you can configure keystores for either the entire container database (CDB) or for individual pluggable databases (PDBs).

Oracle Database supports the following multitenant modes for the management of keystores:

- United mode enables you to configure one keystore for the CDB root and any associated united mode PDBs. United mode operates much the same as how TDE was managed in an multitenant environment in previous releases.
- Isolated mode enables you to create and manage both keystores and TDE master encryption keys in an individual PDB. Different isolated mode PDBs can have different keystore types.

Oracle Database supports isolated PDBs with TDE wallets (wallets) and Oracle Key Vault. The cloud tooling in Oracle Cloud Infrastructure (OCI) and the Oracle Key Management Cloud Service (KMS), do not support isolated PDBs. This includes Oracle Database Exadata Cloud at Customer, Oracle Autonomous Database on Dedicated Exadata Infrastructure, Oracle Exadata Database Service on Cloud@Customer, and Oracle Database@Azure - Oracle Exadata Database Service on Dedicated Infrastructure.

Depending on your site's needs, you can use a mixture of both united mode and isolated mode. For example, if you want most of the PDBs to use one type of a keystore, then you can configure the keystore type in the CDB root (united mode). For the PDBs in this CDB that must use a different type of keystore, then you can configure the PDB itself to use the keystore it needs (isolated mode). The isolated mode setting for the PDB will override the united mode setting for the CDB.

Before you can configure keystores for use in united or isolated mode, you must perform a one-time configuration by using initialization parameters. To configure keystores for united mode and isolated mode, you use the `ADMINISTER KEY MANAGEMENT` statement.

Related Topics

- Configuring United Mode
  United mode enables you to create a common keystore for the CDB and the PDBs for which the keystore is in united mode.
- Configuring Isolated Mode
  Isolated mode enables you to create a keystore for each pluggable database (PDB).

2.7 Transparent Data Encryption Keystore Search Order

The search order for the TDE keystore depends on how you have set either the instance initialization parameters, the `sqlnet.ora` parameters, or the environment variables.

Oracle Database retrieves the keystore by searching in these locations, in the following order:

1. The location set by the `WALLET_ROOT` instance initialization parameter, when the `KEYSTORE_CONFIGURATION` attribute of the `TDE_CONFIGURATION` initialization parameter is set. Oracle recommends that you use this parameter to configure the keystore location.
2. If the `KEYSTORE_CONFIGURATION` attribute of the `TDE_CONFIGURATION` initialization parameter is not set or `WALLET_ROOT` is not set, then the location specified by the `ENCRYPTION_WALLET_LOCATION` setting in the `sqlnet.ora` file.

   Starting with Oracle Database 23ai, the parameter `ENCRYPTION_WALLET_LOCATION` is desupported.
3. If none of these parameters are set, and if the `ORACLE_BASE` environment variable is set, then the `$ORACLE_BASE/admin/`*`db_unique_name`*`/wallet` directory. If `ORACLE_BASE` is not set, then `$ORACLE_HOME/admin/`*`db_unique_name`*`/wallet`.

3
Configuring United Mode

United mode enables you to create a common keystore for the CDB and the PDBs for which the keystore is in united mode.

The keys for the CDB and the PDBs reside in the common keystore.

3.1 About Configuring United Mode

In united mode (the default), the keystore is shared between the CDB root and all PDBs that are configured in united mode. Each united mode PDB has its own set of encryption keys in the shared keystore.

The keys for PDBs having keystore in united mode, can be created from CDB root or from the PDB.

This design enables you to have one keystore to manage the entire CDB environment, enabling the PDBs to share this keystore, but you can customize the behavior of this keystore in the individual united mode PDBs. For example, in a united mode PDB, you can configure a TDE master encryption key for the PDB in the united keystore that you created in the CDB root, open the keystore locally, and close the keystore locally. In order to perform these actions, the keystore in the CDB root must be open.

Before you configure your environment to use united mode or isolated mode, all the PDBs in the CDB environment are considered to be in united mode.

To use united mode, you must follow these general steps:

1. In the CDB root, configure the database to use united mode by setting the `WALLET_ROOT` and `TDE_CONFIGURATION` parameters.

   The `WALLET_ROOT` parameter sets the location for the wallet directory and the `TDE_CONFIGURATION` parameter sets the type of keystore to use.
2. Restart the database after setting the static initialization parameter `WALLET_ROOT`, then set the dynamic initialization parameter `TDE_CONFIGURATION`.
3. In the CDB root, create the keystore, open the keystore, and then create the TDE master encryption key.
4. In each united mode PDB, perform TDE master encryption key tasks as needed, such as opening the keystore locally in the united mode PDB and creating the TDE master encryption key for the PDB. Remember that the keystore is managed by the CDB root, but must contain a TDE master encryption key that is specific to the PDB for the PDB to be able to use TDE.

When you run `ADMINISTER KEY MANAGEMENT` statements in united mode from the CDB root, if the statement accepts the `CONTAINER` clause, and if you set it to `ALL`, then the statement applies only to the CDB root and its associated united mode PDBs. Any PDB that is in isolated mode is not affected.

3.2 Operations That Are Allowed in United Mode

Many `ADMINISTER KEY MANAGEMENT` operations performed in the CDB root apply to keystores and encryption keys in the united mode PDB.

Available United Mode-Related Operations in a CDB Root

Table 3-1 describes the `ADMINISTER KEY MANAGEMENT` operations that you can perform in the CDB root.

Table 3-1 ADMINISTER KEY MANAGEMENT United Mode Operations in a CDB Root

Operation	Syntax	Notes
Creating a keystore	`ADMINISTER KEY MANAGEMENT CREATE KEYSTORE IDENTIFIED BY keystore_password;`	After you create the keystore in the CDB root, by default it is available in the united mode PDBs. Do not include the `CONTAINER` clause.
Opening a keystore	`ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN IDENTIFIED BY [EXTERNAL STORE | keystore_password] [CONTAINER = ALL | CURRENT];`	In this operation, the `EXTERNAL STORE` clause fetches the keystore password from the SSO wallet located in the *`WALLET_ROOT`*`/tde_seps` directory.
Changing a keystore password	`ADMINISTER KEY MANAGEMENT ALTER KEYSTORE PASSWORD IDENTIFIED BY old_keystore_password SET new_keystore_password WITH BACKUP [USING 'backup_identifier'];`	Do not include the `CONTAINER` clause.
Backing up a keystore	`ADMINISTER KEY MANAGEMENT BACKUP KEYSTORE [USING 'backup_identifier'] [FORCE KEYSTORE] IDENTIFIED BY [EXTERNAL STORE | keystore_password] [TO 'keystore_location'];`	Do not include the `CONTAINER` clause.
Closing a keystore without force	`ADMINISTER KEY MANAGEMENT SET KEYSTORE CLOSE [IDENTIFIED BY [EXTERNAL STORE | keystore_password]] [CONTAINER = ALL | CURRENT];`	-
Closing a keystore with force	`ADMINISTER KEY MANAGEMENT FORCE KEYSTORE CLOSE [IDENTIFIED BY [EXTERNAL STORE | keystore_password]] [CONTAINER = ALL | CURRENT];`	-

Table 3-1 (Cont.) ADMINISTER KEY MANAGEMENT United Mode Operations in a CDB Root

Operation	Syntax	Notes
Creating and activating a new TDE master encryption key (rekeying)	`ADMINISTER KEY MANAGEMENT SET [ENCRYPTION] KEY [FORCE KEYSTORE] [USING TAG 'tag_name'] IDENTIFIED BY [EXTERNAL STORE | keystore_password] WITH BACKUP [USING 'backup_identifier'] [CONTAINER = ALL | CURRENT]`	-
Creating a user-defined TDE master encryption key for either now (`SET`) or later on (`CREATE`)	`ADMINISTER KEY MANAGEMENT [SET | CREATE] [ENCRYPTION] KEY 'mkid:mk | mk' [USING ALGORITHM 'algorithm'] [FORCE KEYSTORE] [USING TAG 'tag_name'] IDENTIFIED BY [EXTERNAL STORE | keystore_password] [WITH BACKUP [USING 'backup_identifier']] [CONTAINER = CURRENT];`	-
Activating an existing TDE master encryption key	`ADMINISTER KEY MANAGEMENT USE [ENCRYPTION] KEY 'key_id' IDENTIFIED BY [EXTERNAL STORE | keystore_password] WITH BACKUP [USING 'backup_identifier'];`	Do not include the `CONTAINER` clause.
Tagging a TDE master encryption key	`ADMINISTER KEY MANAGEMENT SET TAG 'tag' FOR 'key_id' IDENTIFIED BY [EXTERNAL STORE | keystore_password ] WITH BACKUP [USING 'backup_identifier'];`	Do not include the `CONTAINER` clause.
Moving a TDE master encryption key to a new keystore	`ADMINISTER KEY MANAGEMENT MOVE [ENCRYPTION] KEYS TO NEW KEYSTORE 'keystore_location1' IDENTIFIED BY keystore1_password FROM [FORCE] KEYSTORE IDENTIFIED BY keystore_password [WITH IDENTIFIER IN { 'key_id' [, 'key_id' ]... | ( subquery ) } ] WITH BACKUP [USING 'backup_identifier'];`	You can only move the master encryption key to a keystore that is within the same container (for example, between keystores in the CDB root or between keystores in the same PDB). You cannot move the master encryption key from a keystore in the CDB root to a keystore in a PDB, and vice versa. Do not include the `CONTAINER` clause.

Available Operations in a United Mode PDB

Table 3-2 describes the `ADMINISTER KEY MANAGEMENT` operations that you can perform in a united mode PDB.

Table 3-2 ADMINISTER KEY MANAGEMENT United Mode PDB Operations

Operation	Syntax	Notes
Opening a keystore	`ADMINISTER KEY MANAGEMENT SET KEYSTORE CLOSE [IDENTIFIED BY EXTERNAL STORE | keystore_password] [CONTAINER = CURRENT];`	In this operation, the `EXTERNAL_STORE` clause uses the password in the TDE SEPS SSO wallet. This wallet is located in the `tde_seps` directory in the `WALLET_ROOT` location.
Closing a keystore without force	`ADMINISTER KEY MANAGEMENT SET KEYSTORE CLOSE [EXTERNAL STORE | keystore_password] [CONTAINER = CURRENT];`	-
Closing a keystore with force	`ADMINISTER KEY MANAGEMENT FORCE KEYSTORE CLOSE IDENTIFIED BY [EXTERNAL STORE | keystore_password] [CONTAINER = CURRENT];`	-
Creating and activating a new TDE master encryption key (rekeying or rotating)	`ADMINISTER KEY MANAGEMENT SET [ENCRYPTION] KEY [FORCE KEYSTORE] [USING TAG 'tag_name'] IDENTIFIED BY EXTERNAL STORE | keystore_password WITH BACKUP [USING 'backup_identifier'] [CONTAINER = CURRENT];`	-
Creating a user-defined TDE master encryption key for use either now (`SET`) or later on (`CREATE`)	`ADMINISTER KEY MANAGEMENT SET | CREATE [ENCRYPTION] KEY 'mkid:mk | mk' [USING ALGORITHM 'algorithm'] [FORCE KEYSTORE] [USING TAG 'tag'] IDENTIFIED BY EXTERNAL STORE | keystore_password WITH BACKUP [USING 'backup_identifier'] [CONTAINER = CURRENT];`	-

Table 3-2 (Cont.) ADMINISTER KEY MANAGEMENT United Mode PDB Operations

Operation	Syntax	Notes
Activating an existing TDE master encryption key	`ADMINISTER KEY MANAGEMENT USE [ENCRYPTION] KEY 'key_id' IDENTIFIED BY EXTERNAL STORE | keystore_password WITH BACKUP [USING 'backup_identifier'];`	Do not include the `CONTAINER` clause.
Tagging a TDE master encryption key	`ADMINISTER KEY MANAGEMENT SET TAG 'tag' FOR 'key_id' [FORCE KEYSTORE] IDENTIFIED BY EXTERNAL STORE | keystore_password WITH BACKUP [USING 'backup_identifier'];`	Do not include the `CONTAINER` clause.
Moving an encryption key to a new keystore	`ADMINISTER KEY MANAGEMENT MOVE [ENCRYPTION] KEYS TO NEW KEYSTORE 'new_keystore_location' IDENTIFIED BY new_keystore_password FROM [FORCE] KEYSTORE IDENTIFIED BY keystore_password [WITH IDENTIFIER IN { 'key_id' [, 'key_id' ]... | ( subquery ) } ] WITH BACKUP [USING 'backup_identifier'];`	Do not include the `CONTAINER` clause.
Moving a key from a united mode keystore in the CDB root to an isolated mode keystore in a PDB	`ADMINISTER KEY MANAGEMENT ISOLATE KEYSTORE IDENTIFIED BY isolated_keystore_password FROM ROOT KEYSTORE [FORCE KEYSTORE] IDENTIFIED BY EXTERNAL STORE | united_keystore_password WITH BACKUP [USING backup_id];`	Do not include the `CONTAINER` clause.

Table 3-2 (Cont.) ADMINISTER KEY MANAGEMENT United Mode PDB Operations

Operation	Syntax	Notes
Using the `FORCE` clause when a clone of a PDB is using the TDE master encryption key that is being isolated; then copying (rather than moving) the TDE master encryption keys from the keystore that is in the CDB root into the isolated mode keystore of the PDB	`ADMINISTER KEY MANAGEMENT FORCE ISOLATE KEYSTORE IDENTIFIED BY` *`isolated_keystore_password`* `FROM ROOT KEYSTORE [FORCE KEYSTORE] IDENTIFIED BY [EXTERNAL STORE | ` *`united_keystore_password`*`] [WITH BACKUP [USING` *`backup_id`*`]];`	-

3.3 Configuring the Keystore Location and Type for United Mode

For united mode, you can configure the keystore location and type by setting parameters and running the `ALTER SYSTEM` statement.

3.3.1 About Configuring the Keystore Location and Type for United Mode

A keystore is a container that stores the TDE master encryption key.

Before you can configure the keystore, you first must define a location for it by setting the static initialization parameter `WALLET_ROOT`. Then, after a database restart, you must set the dynamic initialization parameter `TDE_CONFIGURATION` to instruct the database to retrieve the master encryption key from a TDE wallet, Oracle Key Vault, or Oracle Cloud Interface (OCI) Key Management Service (KMS), according to their documentation. If this setting has not been created, then Oracle Database checks the `sqlnet.ora` file. You can create other TDE wallets, such as copies of the wallet and export files that contain keys, depending on your needs. If you must remove or delete the wallet that you configured in the `WALLET_ROOT` location, then you must do so ***only*** after you copied ***all*** wallets (including backups and auto-login wallets) in the old to the new location. Then you must reset `WALLET_ROOT` to point to the new location of the TDE wallet.

After you configure the keystore location by using the `WALLET_ROOT` and `TDE_CONFIGURATION` parameters, you can log in to the CDB to create and open the TDE wallet, and then set the TDE master encryption key. After you complete these steps, you can begin to encrypt data.

3.3.2 Configuring United Mode with the Initialization Parameter File and ALTER SYSTEM

If your environment relies on server parameter files (`spfile`), then you can set `WALLET_ROOT` and `TDE_CONFIGURATION` using `ALTER SYSTEM SET` with `SCOPE`.

1. Log in to the database server where the CDB root of the Oracle database resides.
2. If necessary, create a wallet directory.

   This directory cannot contain any wallets. Oracle recommends that you create a directory outside of `$ORACLE_HOME` or `$ORACLE_BASE`, to avoid backing up the keystore (wallet or

Oracle Key Vault installation tree) with the encrypted database during an operating system-level backup.

3. Connect to the CDB root as a common user who has been granted the `ALTER SYSTEM` privilege.
4. Run the following `ALTER SYSTEM` statement:

```
ALTER SYSTEM SET WALLET_ROOT='/etc/ORACLE/KEYSTORES/${ORACLE_SID}' SCOPE = SPFILE;
```

5. Restart the CDB.

```
SHUTDOWN IMMEDIATE
STARTUP
```

6. In the CDB root, check the `WALLET_ROOT` setting.

```
SHOW PARAMETER WALLET_ROOT
```

The `VALUE` column of the output should show the absolute path location of the `wallet` directory. For example:

```
wallet_root string /etc/ORACLE/KEYSTORES/finance
```

7. Set the `TDE_CONFIGURATION` dynamic initialization parameter to specify the keystore type, using the following syntax:

```
ALTER SYSTEM SET TDE_CONFIGURATION="KEYSTORE_CONFIGURATION=keystore_type"
SCOPE=BOTH;
```

In this specification:

- *keystore_type* can be one of the following settings:
  - `FILE` configures a TDE keystore.
  - `OKV` configures an Oracle Key Vault keystore.
- *scope_type* sets the type of scope, which in this case is `BOTH`.

To configure a TDE keystore if the server parameter file (`spfile`) is in use:

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

8. Check the `TDE_CONFIGURATION` parameter setting.

```
SHOW PARAMETER TDE_CONFIGURATION
```

The `VALUE` column should show the keystore type, prepended with `KEYSTORE_CONFIGURATION=`.

9. Confirm that the `TDE_CONFIGURATION` parameter was set correctly.

```
SELECT CON_ID, KEYSTORE_MODE FROM V$ENCRYPTION_WALLET;
```

The output should be similar to the following:

```
    CON_ID KEYSTORE
---------- --------
         1 NONE
         2 UNITED
         3 UNITED
         4 UNITED
         5 UNITED
```

The CDB root (`CON_ID 1`) will always be in the `NONE` state, and at this stage, the remaining `CON_ID`s should be set to `UNITED`. PDBs can be either `UNITED` or `ISOLATED`, depending on how you configure them. When you query the `V$ENCRYPTION_WALLET` view, if the `ORA-46691: The value of the KEYSTORE_CONFIGURATION attribute is invalid` error appears, then check the initialization parameter file where you added this setting.

After you configure united mode, you can create keystores and master encryption keys, and when these are configured, you can encrypt data.

3.3.3 Example: Configuring a TDE Wallet When Multiple Databases Share the Same Host

You can configure multiple databases to share the same host by setting the `WALLET_ROOT` parameter.

Because the `WALLET_ROOT` parameter is internal to an Oracle database, you only need to set the parameter to enable multiple databases to share the same host.

For example:

```
ALTER SYSTEM SET WALLET_ROOT = '/etc/ORACLE/KEYSTORES/${ORACLE_SID}' SCOPE = SPFILE;
```

To implement clean separation of individual TDE wallets for each database, add the `ORACLE_SID` (or `ORACLE_UNQNAME` in Oracle Real Application Clusters) into the `WALLET_ROOT` parameter setting.

3.3.4 Example: Configuring a TDE Wallet for an Oracle Automatic Storage Management Disk Group

In an Oracle Real Applications Clusters (Oracle RAC) environment, the `WALLET_ROOT` parameter points to a shared directory in Oracle Automatic Storage Management (ASM) that is accessible from all Oracle RAC instances of that database.

The following example shows you how to set `WALLET_ROOT` and `TDE_CONFIGURATION` for a TDE wallet in ASM:

```
ALTER SYSTEM SET WALLET_ROOT = '+DATA/unique_name_of_database' SCOPE = SPFILE SID = '*';
```

For example, if you set this value to `+DATA/FINRAC`, then the `/tde` directory is automatically generated when you create a TDE wallet.

Related Topics

- Configuring Keystores for Automatic Storage Management
  You can store a TDE wallet on an Automatic Storage Management (ASM) disk group.

3.4 Configuring a TDE Wallet and TDE Master Encryption Key for United Mode

In united mode, the TDE wallet resides in the CDB root but the master keys from this wallet are available for the PDBs that have their TDE wallets in united mode.

3.4.1 About Configuring a TDE Wallet and TDE Master Encryption Key for United Mode

In united mode, the TDE wallet that you create in the CDB root will be accessible by the united mode PDBs.

In general, to configure a united mode TDE wallet after you have enabled united mode, you create and open the TDE wallet in the CDB root, and then create a master encryption key for this TDE wallet. Afterward, you can begin to encrypt data for tables and tablespaces that will be accessible throughout the CDB environment.

The `V$ENCRYPTION_WALLET` dynamic view describes the status and location of the TDE wallet. For example, the following query shows the open-closed status and the TDE wallet location of the CDB root TDE wallet (`CON_ID 1`) and its associated united mode PDBs. The `WRL_PARAMETER` column shows the CDB root TDE wallet location being in the *`WALLET_ROOT`*`/tde` directory.

```
SELECT CON_ID, STATUS, WRL_PARAMETER FROM V$ENCRYPTION_WALLET;

CON_ID STATUS WRL_PARAMETER
------ ------ ----------------------------
     1 OPEN   /app/oracle/wallet/tde/
     2 CLOSED
     3 OPEN
     4 OPEN
     5 OPEN
```

In this output, there is no keystore path listed for the other PDBs in this CDB because these PDBs use the keystore in the CDB root. If any of these PDBs are isolated and you create a keystore in the isolated mode PDB, then when you perform this query, the `WRL_PARAMETER` column will show the keystore path for the isolated mode PDB.

You can hide the TDE wallet password in a secure external password store:

```
ADMINISTER KEY MANAGEMENT ADD SECRET 'TDE_wallet_password'
FOR CLIENT 'TDE_WALLET' INTO [LOCAL]
AUTO_LOGIN KEYSTORE 'WALLET_ROOT/tde_seps';
```

This feature enables you to hide the password from the operating system: it removes the need for storing clear-text keystore passwords in scripts or other tools that can access the database without user intervention, such as overnight batch scripts. The location for this wallet is *`wallet_root`*`/tde_seps`. In a multitenant environment, different PDBs can access this external store location when you run the `ADMINISTER KEY MANAGEMENT` statement using the `IDENTIFIED BY EXTERNAL STORE` clause. This way, you can centrally locate the password and then update it only once in the external store.

3.4.2 Step 1: Create the TDE Wallet

After you have specified a directory location for the TDE wallet, you can create this wallet.

3.4.2.1 About Creating TDE Wallets

There are three different types of TDE wallets.

You can create password-protected TDE wallets, auto-login TDE wallets, and local auto-login TDE wallets.

Be aware that running the query `SELECT * FROM V$ENCRYPTION_WALLET` will automatically open an auto-login TDE wallet. For example, suppose you have a password-protected TDE wallet and an auto-login TDE wallet. If the password-protected TDE wallet is open and you close the password-protected TDE wallet and then query the `V$ENCRYPTION_WALLET` view, then the output will indicate that a TDE wallet is open. However, this is because `V$ENCRYPTION_WALLET` opened up the auto-login TDE wallet and then displayed the status of the auto-login wallet.

Related Topics

- Types of Keystores
  Oracle Database supports TDE wallets, Oracle Key Vault, and Oracle Cloud Infrastructure (OCI) key management systems (KMS).

3.4.2.2 Creating a Password-Protected TDE Wallet

A password-protected TDE wallet requires a password, which is used to protect the TDE master keys. In united mode, you must create the TDE wallet in the CDB root.

After you create this TDE wallet in the CDB root, it becomes available for all united PDBs, but not for isolated PDBs.

A TDE wallet can only contain TDE-related security objects, and no security objects used by other database or application components. In addition, do not use orapki or mkstore to create password-protected or (local) auto-open TDE wallets. Instead, use the `ADMINISTER KEY MANAGEMENT statement`.

1. Connect to the CDB root as a common user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.
2. Use the `SHOW PARAMETER` command to confirm that `WALLET_ROOT` is set, and `TDE_CONFIGURATION` is set to `KEYSTORE_CONFIGURATION=FILE`.
3. Run the `ADMINISTER KEY MANAGEMENT` SQL statement to create the TDE wallet using the following syntax:

   ```
   ADMINISTER KEY MANAGEMENT CREATE KEYSTORE
   IDENTIFIED BY TDE_wallet_password;
   ```

   The `/tde` directory is automatically created under `WALLET_ROOT`, if it does not already exist. In Oracle Automatic Storage Management (Oracle ASM), for the `/tde` directory to be automatically created if it does not already exist, `WALLET_ROOT` must point to `+DATA/ORACLE_UNQNAME`.

   In this specification, *`TDE_wallet_password`* is the password of the TDE wallet that you, the security administrator, creates.

   For example, to create the TDE wallet in the *`WALLET_ROOT`*`/tde` directory:

   ```
   ADMINISTER KEY MANAGEMENT CREATE KEYSTORE IDENTIFIED BY TDE_wallet_password;

   keystore altered.
   ```

   After you run this statement, the `ewallet.p12` file, which is the TDE wallet, appears in the TDE wallet location.

After you complete these steps, the `ewallet.p12` file, which contains the TDE wallet, appears in the designated TDE wallet location. For example, if you had set the `WALLET_ROOT` parameter to `/etc/ORACLE/KEYSTORES/${ORACLE_SID}` and the `TDE_CONFIGURATION` parameter to `FILE` (for

TDE, which creates a `tde` directory in the wallet root location), then the TDE wallet will be created in the `/etc/ORACLE/KEYSTORES/${ORACLE_SID}/tde` directory. The name of the TDE wallet is `ewallet.p12`.

Related Topics

- Configuring an External Store for a Keystore Password
  An external store for a keystore password stores the keystore password in a centrally accessed and managed location.

3.4.2.3 Creating an Auto-Login or a Local Auto-Login TDE Wallet

As an alternative to password-protected TDE wallets, you can create either an auto-login or local auto-login TDE wallet.

Both of these TDE wallets have system-generated passwords. They are also PKCS#12-based files. The auto-login TDE wallet can be opened from different computers from the computer where this TDE wallet resides, but the local auto-login TDE wallet can only be opened from the computer on which it was created. Both the auto-login and local auto-login TDE wallets are created from the password-protected TDE wallets. Creating any of them does not require database downtime.

1. Connect to the CDB root as a common user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.
2. Use the `SHOW PARAMETER` command to confirm that `WALLET_ROOT` is set, and `TDE_CONFIGURATION` is set to `KEYSTORE_CONFIGURATION=FILE`.
3. Create the auto-login or local auto-login TDE wallet by using the following syntax:

   ```
   ADMINISTER KEY MANAGEMENT CREATE [LOCAL] AUTO_LOGIN KEYSTORE
   FROM KEYSTORE IDENTIFIED BY TDE_wallet_password;
   ```

   In this specification:

   - `LOCAL` enables you to create a local auto-login TDE wallet. Otherwise, omit this clause if you want the TDE wallet to be accessible by other computers. `LOCAL` creates a local auto-login wallet file, `cwallet.sso`, and this wallet will be tied to the host on which it was created. For an Oracle Real Application Clusters (Oracle RAC) environment, omit the `LOCAL` keyword, because each Oracle RAC node has a different host name. If you configure a local auto-login wallet for the Oracle RAC instance, then only the first Oracle RAC node, where the `cwallet.sso` file was created, would be able to access the TDE wallet. If you try to open the TDE wallet from another node instead of from that first node, there would be a problem auto-opening `cwallet.sso`, and so it would result in a failure to auto-open the TDE wallet. This restriction applies if you are using a shared location to hold the `cwallet.sso` file for the Oracle RAC cluster, because using `LOCAL` only works if you have a separate `cwallet.sso` file (containing the same credentials) on each node of the Oracle RAC environment.
   - *`TDE_wallet_password`* is the password of the TDE wallet from which you want to create.

   For example, to create an auto-login TDE wallet of the password-protected TDE wallet that is located in the *`<WALLET_ROOT>`*`/tde` directory:

   ```
   ADMINISTER KEY MANAGEMENT CREATE AUTO_LOGIN KEYSTORE
   FROM KEYSTORE IDENTIFIED BY password;

   keystore altered.
   ```

After you run this statement, the `cwallet.sso` file appears in the TDE wallet location. The `ewallet.p12` file is the password-protected wallet.

Follow these guidelines:

- Do not remove the `PKCS#12` wallet (`ewallet.p12` file) after you create the auto-login TDE wallet (`.sso` file). You must have the `PKCS#12` wallet to regenerate or rekey the TDE master encryption key in the future.
- Remember that Transparent Data Encryption uses an auto login TDE wallet only if it is available at the correct location (*`WALLET_ROOT`*`/tde`), and the SQL statement to open an encrypted TDE wallet has not already been run. If you have the `ENCRYPTION_WALLET_LOCATION` parameter set, then be aware this parameter is deprecated. Oracle recommends that you use the `WALLET_ROOT` static initialization parameter and `TDE_CONFIGURATION` dynamic initialization parameter instead.

3.4.3 Step 2: Open the TDE Wallet

Depending on the type of TDE wallet you create, you must manually open the wallet before you can use it.

3.4.3.1 About Opening TDE Wallets

A password-protected TDE wallet must be open before any TDE master encryption keys can be created or accessed in the wallet.

Many Transparent Data Encryption operations require the TDE wallet to be open. There are two ways that you can open the TDE wallet:

- Manually open the wallet by issuing the `ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN` statement. Afterward, you can perform the operation.
- Include the `FORCE KEYSTORE` clause in the `ADMINISTER KEY MANAGEMENT` statement that is used to perform the operation. `FORCE KEYSTORE` temporarily opens the TDE wallet for the duration of the operation, and when the operation completes, the TDE wallet is closed again. `FORCE KEYSTORE` is useful for situations when the database is heavily loaded. In this scenario, because of concurrent access to encrypted objects in the database, the auto-login TDE wallet continues to open immediately after it has been closed but before a user has had chance to open the password-based TDE wallet.

TDE wallets can be in the following states: open, closed, open but with no master encryption key, open but with an unknown master encryption key, undefined, or not available (that is, not present in the `WALLET_ROOT/tde` location).

After you manually open a TDE wallet, it remains open until you manually close it. Each time you restart a PDB or CDB, you must manually open the password TDE wallet to reenable encryption and decryption operations.

You can check the status of whether a TDE wallet is open or not by querying the `STATUS` column of the `V$ENCRYPTION_WALLET` view.

Related Topics

- Performing Operations That Require a Keystore Password
  Many `ADMINISTER KEY MANAGEMENT` operations require access to a keystore password, for both TDE wallets and external keystores.
- How Keystore Open and Close Operations Work in United Mode
  You should be aware of how keystore open and close operations work in united mode.

3.4.3.2 Opening the TDE Wallet in a United Mode PDB

To open a TDE wallet in united mode, you must use the `ADMINISTER KEY MANAGEMENT` statement with the `SET KEYSTORE OPEN` clause.

1. Connect to the CDB root as a common user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.
2. Open the TDE wallet in the CDB root.

   For example:

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

   keystore altered.
   ```

   If the CDB is configured with the TDE wallet password stored in *`WALLET_ROOT`*`/tde_seps` and has a (local) auto-login TDE wallet in *`WALLET_ROOT`*`/tde`, you must include the `FORCE KEYSTORE` clause and the `IDENTIFIED BY EXTERNAL STORE` clause in the `ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN` statement, as follows:

   ```
   ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN
   FORCE KEYSTORE
   IDENTIFIED BY EXTERNAL STORE;

   keystore altered.
   ```

   If the `WALLET_ROOT` parameter has been set, then Oracle Database finds the external store in the *`WALLET_ROOT`*`/tde_seps` directory.
3. Ensure that the PDB in which you want to open the TDE wallet is in `READ WRITE` mode.

   For example:

   ```
   SHOW PDBS

   CON_ID CON_NAME   OPEN MODE    RESTRICTED
   ------ ---------- ------------ -----------
   2      PDB$SEED   READ ONLY    NO
   3      CDB1_PDB1  READ WRITE   NO
   ```

   If any PDB has an `OPEN MODE` value that is different from `READ WRITE`, then run the following statement to open the PDB, which will set it to `READ WRITE` mode:

   ```
   ALTER PLUGGABLE DATABASE CDB1_PDB1 OPEN;
   ```

   Now the TDE wallet can be opened in both the CDB root and the PDB.
4. Connect to the PDB.
5. Run the `ADMINISTER KEY MANAGEMENT` statement to open the TDE wallet.

   ```
   ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN
   IDENTIFIED BY TDE_wallet_password | EXTERNAL STORE;
   keystore altered.
   ```

   To open a password-protected TDE wallet when a (local) auto-login TDE wallet is open, specify the `FORCE KEYSTORE` clause as follows.

   ```
   ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN
   FORCE KEYSTORE
   ```

```
IDENTIFIED BY TDE_wallet_password | EXTERNAL STORE;
keystore altered.
```

`FORCE KEYSTORE` is also useful for databases that are heavily loaded. The `IDENTIFIED BY EXTERNAL STORE` clause is included in the statement because the TDE wallet credentials exist in an external store. This enables the password-protected TDE wallet to be opened without specifying the TDE wallet password within the statement itself.

If the `WALLET_ROOT` parameter has been set in the CDB, then united PDBs find the external store in the *`WALLET_ROOT`*`/tde_seps` directory.

6. Confirm that the TDE wallet is open.

```
SELECT STATUS FROM V$ENCRYPTION_WALLET;
```

Note that if the TDE wallet is open but you have not created a TDE master encryption key yet, the `STATUS` column of the `V$ENCRYPTION_WALLET` view reminds you with an `OPEN_NO_MASTER_KEY` status.

3.4.4 Step 3: Set the TDE Master Encryption Key in the TDE Wallet

Once the TDE wallet is open, you can set a TDE master encryption key for it.

3.4.4.1 About Setting the TDE Wallet TDE Master Encryption Key

The TDE master encryption key is stored in the TDE wallet.

The TDE master encryption key protects the TDE table keys and tablespace encryption keys. By default, the TDE master encryption key is a key that TDE generates. You can find if a TDE wallet has no TDE master encryption key set or an unknown TDE master encryption key by querying the `STATUS` column of the `V$ENCRYPTION_WALLET` view.

You can import a master encryption key (bring your own key (BYOK)), that was created outside of the database, into the TDE wallet. You can either set a key for immediate use, using `ADMINISTER KEY MANAGEMENT SET`, or create a key using `ADMINISTER KEY MANAGEMENT CREATE KEY` for later use, and activate it with the `ADMINISTER KEY MANAGEMENT USE KEY` statement.

Related Topics

- Creating User-Defined TDE Master Encryption Keys
  You can create a user-defined TDE master encryption key outside the database by generating a TDE master encryption key ID.
- Creating TDE Master Encryption Keys for Later Use in United Mode
  You can create a TDE master encryption key that can be activated at a later date.

3.4.4.2 Setting the TDE Master Encryption Key in the United Mode TDE Wallet

To set the TDE master encryption key in the TDE wallet when the PDB is configured in united mode, use the `ADMINISTER KEY MANAGEMENT` statement with the `SET KEY` clause.

1. Connect to the CDB root as a common user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.
2. Ensure that the CDB is open in `READ WRITE` mode.

   You can set the TDE master encryption key if `OPEN_MODE` is set to `READ WRITE`. To find the status, query the `OPEN_MODE` column of the `V$DATABASE` dynamic view. (If you cannot access this view, then connect as `SYSDBA` and try the query again. In order to connect as `SYSKM` for this type of query, you must create a password file for it.)

3. Run the `ADMINISTER KEY MANAGEMENT SET ENCRYPTION KEY` statement to set the TDE master encryption key in the TDE wallet.

   Use the following syntax:

   ```
   ADMINISTER KEY MANAGEMENT SET KEY
   [USING TAG 'tag']
   [FORCE KEYSTORE]
   IDENTIFIED BY EXTERNAL STORE | TDE_wallet_password
   WITH BACKUP [USING 'backup_identifier'];
   ```

   In this specification:

   - *tag* is the associated attributes and information that you define. Enclose this setting in single quotation marks (`' '`).
   - `FORCE KEYSTORE` should be included if the TDE wallet is closed. This automatically opens the keystore before setting the TDE master encryption key. The `FORCE KEYSTORE` clause also switches over to opening the password-protected TDE wallet when an auto-login TDE wallet is configured and is currently open.
   - `IDENTIFIED BY` specifies the TDE wallet password. Alternatively, if the TDE wallet password is in an external store, you can use the `IDENTIFIED BY EXTERNAL STORE` clause.
   - `WITH BACKUP` creates a backup of the TDE wallet. You must use this option for password-protected TDE wallets. Optionally, you can use the `USING` clause to add a brief description of the backup. Enclose this description in single quotation marks (`' '`). This identifier is appended to the named TDE wallet file (for example, `ewallet_`*time_stamp*`_emp_key_backup.p12`, with `emp_key_backup` being the backup identifier). Follow the file naming conventions that your operating system uses.

   For example, if the TDE wallet is password-protected and open, and you want to set the TDE master encryption key in the current container:

   ```
   ADMINISTER KEY MANAGEMENT SET KEY
   IDENTIFIED BY EXTERNAL STORE | TDE_wallet_password
   WITH BACKUP USING 'emp_key_backup';
   ```

   If the TDE wallet is closed:

   ```
   ADMINISTER KEY MANAGEMENT SET KEY
   [USING TAG 'tag']
   FORCE KEYSTORE
   IDENTIFIED BY EXTERNAL STORE | TDE_wallet_password
   WITH BACKUP USING 'emp_key_backup';
   ```

4. Confirm that the TDE master encryption key is set.

   ```
   SELECT MASTERKEY_ACTIVATED FROM V$DATABASE_KEY_INFO;
   ```

   The output should be `YES`.

Related Topics

- About Setting the TDE Wallet TDE Master Encryption Key
  The TDE master encryption key is stored in the TDE wallet.

3.4.5 Step 4: Encrypt Your Data in United Mode

Now that you have completed the configuration, you can begin to encrypt data.

Related Topics

- Encrypting Columns in Tables
  You can use Transparent Data Encryption to encrypt individual columns in database tables.
- Encryption Conversions for Tablespaces and Databases
  You can perform encryption operations on both offline and online tablespaces and databases.

3.5 Operations That Are Not Allowed in a United Mode PDB

`ADMINISTER KEY MANAGEMENT` operations that are not allowed in a united mode PDB can be performed in the CDB root.

These operations are as follows:

- Keystore operations:
  - Performing merge operations on keystores
  - Exporting a keystore
  - Importing a keystore
  - Migrating a keystore
  - Reverse-migrating a keystore
  - Moving the keys of a keystore that is in the CDB root into the keystores of a PDB
  - Moving the keys from a PDB into a united mode keystore that is in the CDB root
- Encryption key operations:
  - Using the `CONTAINER = ALL` clause to create a new TDE master encryption key for later user in each pluggable database (PDB)
- Client secret operations:
  - Adding client secrets
  - Updating client secrets
  - Deleting client secrets

3.6 Configuring a Container Database with United Mode PDBs for Oracle Key Vault

The TDE master keys of all united mode PDBs and the CDB reside in the same virtual wallet in Oracle Key Vault.

3.6.1 About Configuring a Container Database with United Mode PDBs for Oracle Key Vault

Oracle Key Vault is a fault-tolerant, scalable, and continuously available key and secrets management platform that was purpose-built for TDE master key management even for the largest, and most diverse, Oracle database deployments.

External keystores are external to an Oracle database. Oracle Database can interface with external keystores but cannot manipulate them outside of the Oracle interface. The Oracle

database can request the external keystore to create a key but it cannot define how this key is stored in an external database. Examples of external keystores are Oracle Key Vault keystores. Supported external keystores are Oracle Key Vault and the Oracle Cloud Infrastructure Vault. (Conversely, for TDE wallets that are created using TDE, Oracle Database has full control: that is, you can use SQL statements to manipulate this type of keystore.)

To configure an external keystore, you must first define the keystore type in the `TDE_CONFIGURATION` parameter setting, configure and open the external keystore, and then set the first TDE master encryption key in the external keystore. In short, there is one external keystore per database, and the database locates this keystore by checking the keystore type that you define in the `TDE_CONFIGURATION` parameter.

3.6.2 About Configuring a Container Database with United Mode PDBs for Oracle Key Vault

In united mode, you can configure Oracle Key Vault by setting the `WALLET_ROOT` and `TDE_CONFIGURATION` parameters in the container database (CDB).

Oracle recommends that you set the parameters `WALLET_ROOT` and `TDE_CONFIGURATION` for new deployments. Alternatively, you can migrate from the old configuration in the `sqlnet.ora file` to the new configuration with `WALLET_ROOT` and `TDE_CONFIGURATION` at your earliest convenience (for example, the next time you apply a quarterly bundle patch).

United Mode is the default TDE setup that is used in Oracle Database release 12.1.0.2 and later with the TDE configuration in `sqlnet.ora`. In Oracle Database release 18c and later, TDE configuration in `sqlnet.ora` is deprecated. You must first set the static initialization parameter `WALLET_ROOT` to an existing directory; for this change to be picked up, a database restart is necessary. After the restart, set the `KEYSTORE_CONFIGURATION` attribute of the dynamic `TDE_CONFIGURATION` parameter to `OKV` (for a password-protected connection into Oracle Key Vault), or `OKV|FILE` for an auto-open connection into Oracle Key Vault, and then open the configured external keystore, and then set the TDE master encryption keys. After you complete these tasks, you can begin to encrypt data in your database.

3.6.3 Step 1: Configure Oracle Key Vault for United Mode

You can configure Oracle Key Vault for united mode PDBs by setting the `WALLET_ROOT` and `TDE_CONFIGURATION` parameters in the container database (CDB).

1. If the `WALLET_ROOT` parameter is set, then install the Oracle Key Vault client software into the `WALLET_ROOT/okv` directory.
2. Log in to the database instance as a user who has been granted the `ALTER SYSTEM` administrative privilege.
3. Set the `TDE_CONFIGURATION` dynamic initialization parameter to specify the keystore type by using the following syntax:

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

   In this specification:

   - *`keystore_type`* is `OKV`, to configure an Oracle Key Vault keystore.

   For example, to configure your database to use Oracle Key Vault:

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

3.6.4 Step 2: Open the Connection to Oracle Key Vault

After you have configured the database to use Oracle Key Vault for TDE key management, you must open the connection to Oracle Key Vault before you can use it.

3.6.4.1 About Opening the Connection to Oracle Key Vault

You must open the connection to Oracle Key Vault so that it is accessible to the database before you can perform any encryption or decryption.

If a recovery operation is needed on your database (for example, if the database was not cleanly shut down, and has an encrypted tablespace that needs recovery), then you must open the connection to Oracle Key Vault before you can open the database itself.

There are two ways that you can open the Oracle Key Vault connection:

- Manually open the keystore by issuing the `ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN` statement. Afterward, you can perform the operation.
- Include the `FORCE KEYSTORE` clause in the `ADMINISTER KEY MANAGEMENT` statement. `FORCE KEYSTORE` temporarily opens the keystore for the duration of the operation, and when the operation completes, the keystore is closed again. `FORCE KEYSTORE` is useful for situations when the database is heavily loaded. In this scenario, because of concurrent access to encrypted objects in the database, the auto-login keystore continues to open immediately after it has been closed but before a user has had a chance to open the password-based keystore.

To check the status of the keystore, query the `STATUS` column of the `V$ENCRYPTION_WALLET` view. Keystores can be in the following states: `CLOSED`, `NOT_AVAILABLE` (that is, not present in the `WALLET_ROOT` location), `OPEN`, `OPEN_NO_MASTER_KEY`, `OPEN_UNKNOWN_MASTER_KEY_STATUS`.

Be aware that for external keystores, if the database is in the mounted state, then it cannot check if the master key is set because the data dictionary is not available. In this situation, the status will be `OPEN_UNKNOWN_MASTER_KEY_STATUS`.

Related Topics

- How Keystore Open and Close Operations Work in United Mode
  You should be aware of how keystore open and close operations work in united mode.

3.6.4.2 Opening the Oracle Key Vault Connection in a United Mode PDB

To open the Oracle Key Vault connection in united mode, you must use the `ADMINISTER KEY MANAGEMENT` statement with the `SET KEYSTORE OPEN` clause.

1. Connect to the CDB root as a common user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.
2. Open the keystore in the CDB root by using the following syntax.

```
ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN
IDENTIFIED BY EXTERNAL STORE | "Oracle_Key_Vault_password"
[CONTAINER = ALL | CURRENT];
```

   In this specification:

   - `IDENTIFIED BY` can be one of the following settings:

  - `EXTERNAL STORE` uses the keystore password stored in the external store to perform the keystore operation.
  - *`Oracle_Key_Vault_password`* is the Oracle Key Vault password that was giving during the Oracle Key Vault client installation. If no password was given, then the password in the `ADMINISTER KEY MANAGEMENT` statement becomes `NULL`. Enclose this password in double quotation marks.
- `CONTAINER` specifies the scope in which to open the Oracle Key Vault connection.
  - `ALL` opens the connection to Oracle Key Vault for the root container and all open united PDBs.
  - `CURRENT` opens the connection to Oracle Key Vault for the root container.

If the password contains a semi-colon (`;`), then enclose the password in double quotation marks.

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

3. Ensure that the PDB in which you want to open the keystore is in `READ WRITE` mode.
4. Connect to the PDB and run the `ADMINISTER KEY MANAGEMENT` statement to open the keystore.

   For example:

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

5. Confirm that the keystore is open.

   ```
   SELECT STATUS FROM V$ENCRYPTION_WALLET;
   ```

6. Repeat this procedure each time you restart the PDB.

3.6.5 Step 3: Set the TDE Master Encryption Key in Oracle Key Vault

After you have opened the connection to Oracle Key Vault, you are ready to set the TDE master encryption key.

3.6.5.1 About Setting the External Keystore TDE Master Encryption Key

You must create a TDE master encryption key that is stored inside the external keystore.

Oracle Database uses the master encryption key from Oracle Key Vault or Oracle Cloud Infrastructure (OCI) Key Management Service (KMS) to encrypt or decrypt TDE table keys or tablespace encryption keys (data encryption keys) inside the database.

If you have not previously configured TDE with a wallet, then you must set the master encryption key in Oracle Key Vault or OCI KMS. If you have already configured TDE with a wallet, then you must migrate the database to Oracle Key Vault or OCI KMS.

Along with the current master encryption key, all TDE keystores (TDE wallet, Oracle Key Vault, and OCI KMS) maintain historical master encryption keys that are generated after every re-key operation that rekeys the master encryption key. These historical master keys help to restore Oracle Database backups that were taken previously using one of the master encryption keys. Only Oracle Key Vault allows you to upload all historical key from a TDE wallet before migrating the database to use Oracle Key Vault. After migration to Oracle Key Vault, the TDE wallet can be deleted, which satisfies security regulation that mandate that encryption keys cannot reside on the encrypting server.

3.6.5.2 Heartbeat Batch Size for External Keystores

You can control the size of the batch of heartbeats issued during each heartbeat period.

When a PDB is configured to use an external key manager, the GEN0 background process must perform a heartbeat request on behalf of the PDB to the external key manager. This background process ensures that the external key manager is available and that the TDE master encryption key of the PDB is available from the external key manager and can be used for both encryption and decryption. The GEN0 background process must complete this request within the heartbeat period (which defaults to three seconds).

When a very large number of PDBs (for example, 1000) are configured to use an external key manager, you can configure the HEARTBEAT_BATCH_SIZE database instance initialization parameter to batch heartbeats and thereby mitigate the possibility of the hang analyzer mistakenly flagging the GEN0 process as being stalled when there was not enough time for it to perform a heartbeat for each PDB within the allotted heartbeat period.

By setting the heartbeat batch size, you can stagger the heartbeats across batches of PDBs to ensure that for each batch a heartbeat can be completed for each PDB within the batch during the heartbeat period, and also ensure that PDB master encryption keys can be reliably fetched from an Oracle Key Vault server and cached in the Oracle Key Vault persistent cache. The HEARTBEAT_BATCH_SIZE parameter configures the size of the batch of heartbeats sent per heartbeat period to the external key manager. The value must be between 2 and 100 and it defaults to 5. The default duration of the heartbeat period is three seconds.

For example, if 500 PDBs are configured and are using Oracle Key Vault, the usual time taken by GEN0 to perform a heartbeat on behalf of a single PDB is less than half a second. In addition, assume that the CDB$ROOT has been configured to use an external key manager such as Oracle Key Vault (OKV). Therefore, it should generally be possible to send five heartbeats (one for the CDB$ROOT and four for a four-PDB batch) in a single batch within every three-second heartbeat period.

Even though the HEARTBEAT_BATCH_SIZE parameter configures the number of heartbeats sent in a batch, if the CDB$ROOT is configured to use an external key manager, then each heartbeat batch must include a heartbeat for the CDB$ROOT. The minimum value of the HEARTBEAT_BATCH_SIZE parameter is 2 and its maximum value is 100. When the CDB$ROOT is configured to use an external key manager, then each batch of heartbeats includes one heartbeat for the CDB$ROOT. This is why the minimum batch size is two: one must be reserved for the CDB$ROOT, because it might be configured to use an external key manager.

For example, suppose you set the HEARTBEAT_BATCH_SIZE parameter as follows:

```
ALTER SYSTEM SET HEARTBEAT_BATCH_SIZE=3 SCOPE=BOTH SID='*';
```

Each iteration corresponds to one GEN0 three-second heartbeat period.

Example 1: Setting the Heartbeat for Containers That Are Configured to Use Oracle Key Vault

Suppose the container list is 1 2 3 4 5 6 7 8 9 10, with all containers configured to use Oracle Key Vault (OKV). The iterations are as follows:

- Iteration 1: batch consists of containers: 1 2 3
- Iteration 2: batch consists of containers: 1 4 5
- Iteration 3: batch consists of containers: 1 6 7

- Iteration 4: batch consists of containers: 1 8 9
- Iteration 5: batch consists of containers: 1 10
- Repeat this cycle.

Example 2: Setting the Heartbeat for Isolated PDBs with Different Keystores (Root Container in Oracle Key Vault)

In this example, the container list is 1 2 3 4 5 6 7 8 9 10, with only odd-numbered PDBs configured to use OKV, and the even-numbered PDBs configured to use a TDE wallet (`FILE`).

- Iteration 1: batch consists of containers: 1 3 5
- Iteration 2: batch consists of containers: 1 7 9
- Iteration 3: batch consists of containers: 1
- Repeat this cycle.

Example 3: Setting the Heartbeat for Isolated PDBs with Different Keystores (Root Container in TDE Wallet)

Assume that the container list is 1 2 3 4 5 6 7 8 9 10, with only even-numbered container numbers configured to use Oracle Key Vault, and the odd-numbered containers configured to use `FILE`. In the following example, there is no heartbeat for the `CDB$ROOT`, because it is configured to use `FILE`.

- Iteration 1: batch consists of containers: 2 4 6
- Iteration 2: batch consists of containers: 8 10
- Repeat this cycle.

Related Topics

- Setting the TDE Master Encryption Key for United Mode PDBs in an External Keystore
  To set the TDE master encryption key in the keystore when the PDB is configured in united mode, use the `ADMINISTER KEY MANAGEMENT` statement with the `SET KEY` clause.
- Setting the TDE Master Encryption Key in the Isolated Mode External Keystore
  You should complete this procedure if you have not previously configured an external keystore for Transparent Data Encryption.

3.6.5.3 Setting the TDE Master Encryption Key for United Mode PDBs in an External Keystore

To set the TDE master encryption key in the keystore when the PDB is configured in united mode, use the `ADMINISTER KEY MANAGEMENT` statement with the `SET KEY` clause.

1. Connect to the CDB root as a common user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.
2. Ensure that the database is open in `READ WRITE` mode.

   You can set the master encryption key if `OPEN_MODE` is set to `READ WRITE`. To find the status, for a non-multitenant environment, query the `OPEN_MODE` column of the `V$DATABASE` dynamic view. If you are in a multitenant environment, then run the `show pdbs` command.
3. To enable or disable in-memory caching of master encryption keys, set the `TDE_KEY_CACHE` initialization parameter.

   This optional setting is only available for databases in Oracle Cloud Infrastructure (OCI) that use the OCI Key Management Service (KMS) for key management.

Enabling in-memory caching of master encryption keys helps to reduce the dependency on the OCI KMS during the decryption of data encryption keys. By having the master encryption key local to the database, you can improve the database availability by avoiding the failures that can happen because of intermittent network issues if the calls were made to the key server instead. A setting of `TRUE` enables in-memory caching; `FALSE` disables it.

```
ALTER SYSTEM SET TDE_KEY_CACHE = TRUE SCOPE=BOTH SID='*';
```

4. To configure the heartbeat batch size, set the `HEARTBEAT_BATCH_SIZE` initialization parameter.

   The `HEARTBEAT_BATCH_SIZE` parameter configures the size of the "batch of heartbeats" sent per heartbeat period to the external key manager. Enter a value between 2 and 100. The default value is 5. The default duration of the heartbeat period is three seconds. By setting the heartbeat batch size, you can stagger the heartbeats across batches of PDBs to ensure that for each batch a heartbeat can be completed for each PDB within the batch during the heartbeat period, and also ensure that the TDE master encryption key of the PDB can be reliably fetched from an Oracle Key Vault server and cached in the Oracle Key Vault persistent cache. (See Heartbeat Batch Size for External Keystores for details about how `HEARTBEAT_BATCH_SIZE` works.)

   ```
   ALTER SYSTEM SET HEARTBEAT_BATCH_SIZE=3 SCOPE=BOTH SID='*';
   ```

5. Run the `ADMINISTER KEY MANAGEMENT SET ENCRYPTION KEY` statement to set the TDE master encryption key in the keystore.

   ```
   ADMINISTER KEY MANAGEMENT SET KEY
   [USING TAG 'tag']
   [FORCE KEYSTORE]
   IDENTIFIED BY [EXTERNAL STORE | "external_key_manager_password"];
   ```

   In this specification:

   - *tag* is the associated attributes and information that you define. Enclose this setting in single quotation marks (`' '`).
   - `FORCE KEYSTORE` temporarily opens the password-protected TDE wallet for this operation. You must open the TDE wallet for this operation.
   - `IDENTIFIED BY` can be one of the following settings:
     - `EXTERNAL STORE` uses the keystore password stored in the external store to perform the keystore operation.
     - *external_key_manager_password* is for an external keystore manager, which can be Oracle Key Vault or OCI Vault - Key Management. Enclose this password in double quotation marks. For Oracle Key Vault, enter the password that was given during the Oracle Key Vault client installation. If at that time no password was given, then the password in the `ADMINISTER KEY MANAGEMENT` statement becomes `NULL`.

   For example:

   ```
   ADMINISTER KEY MANAGEMENT SET KEY
   FORCE KEYSTORE
   IDENTIFIED BY "external_key_manager_password";

   keystore altered.
   ```

6. Confirm that the TDE master encryption key is set.

```
SELECT MASTERKEY_ACTIVATED FROM V$DATABASE_KEY_INFO;
```

The output should be `YES`.

3.6.5.4 Migration of an Encrypted Database from a TDE Wallet to Oracle Key Vault or OCI KMS

To switch from a TDE wallet to centralized key management with Oracle Key Vault or Oracle Cloud Infrastructure (OCI) Key Management Service (KMS), after you upload all current and retired TDE master keys you must migrate the database from the TDE wallet to Oracle Key Vault or OCI KMS.

Tools such as Oracle Data Pump and Oracle Recovery Manager require access to the old TDE wallet to perform decryption and encryption operations on data exported or backed up using the TDE wallet. Along with the current master encryption key, Oracle keystores maintain historical master encryption keys that are generated after every re-key operation that rotates the master encryption key. These historical master encryption keys help to restore Oracle database backups that were taken previously using one of the historical master encryption keys.

Related Topics

- Migrating from a TDE Wallet to Oracle Key Vault
  You can migrate between password-protected TDE wallets and external keystores in Oracle Key Vault.

3.6.6 Step 4: Encrypt Your Data in United Mode

Now that you have completed the configuration for an external keystore or for an Oracle Key Vault keystore, you can begin to encrypt data.

Related Topics

- Encrypting Columns in Tables
  You can use Transparent Data Encryption to encrypt individual columns in database tables.
- Encryption Conversions for Tablespaces and Databases
  You can perform encryption operations on both offline and online tablespaces and databases.
- *Oracle Key Vault Administrator's Guide*

4
Configuring Isolated Mode

Isolated mode enables you to create a keystore for each pluggable database (PDB).

4.1 About Configuring Isolated Mode

In isolated mode, where a pluggable database (PDB) has its own keystore and keystore password, you manage the keystore and its TDE master encryption keys from the PDB only.

Similar to united mode, you must first configure a PDB to use isolated mode by setting the `WALLET_ROOT` and `TDE_CONFIGURATION` parameters. After you set these parameters, you can create and manage the keystore from the PDB. In this way, you can have the following scenario:

- United mode PDBs inherit the TDE configuration from the root container. For example, the keystore that you create in the CDB root will be used by the root's associated united mode PDBs.
- The PDBs that are configured in isolated mode are allowed to independently create and manage their own keystore. An isolated mode PDB can have its own keystore, independent of the keystore of the CDB root.

This scenario is useful in cases where you have many PDBs that must use one type of keystore, but you have a few PDBs that must use a different type. By different types of keystores, this refers to either TDE wallet or to one of the external keystores that Oracle supports (for example, Oracle Key Vault or Cloud Key Management Service). You cannot have a mixture of different external keystore types in one CDB environment because the Oracle server can load only one PKCS#11 vendor library. If necessary, you can configure these PDBs in isolated mode so that each PDB can use its own keystore.

> **Note:**
>
> Oracle Cloud Infrastructure (OCI) cloud tooling does not support isolated PDBs. This non-support applies to Oracle Base Database Service (BaseDB), Oracle Exadata Database Service on Dedicated Infrastructure (ExaDB-D), ADB-D, ExaDB-D@Azure, Oracle Exadata Database Service on Cloud@Customer(ExaDB-C@C) , Oracle Autonomous Database on Dedicated Exadata Infrastructure (ADB-C@C), even when Oracle Key Vault provides key management for those database deployments.

4.2 Operations That Are Allowed in Isolated Mode

You can perform many `ADMINISTER KEY MANAGEMENT` operations in isolated mode.

These operations include creating, backing up, opening keystores; changing keystore passwords, merging keystores, closing keystores; creating, activating, tagging, moving, exporting, importing, and migrating encryption keys; and adding, updating, and deleting client secrets.

Table 4-1 describes the `ADMINISTER KEY MANAGEMENT` operations that you can perform in an isolated mode PDB.

Table 4-1 ADMINISTER KEY MANAGEMENT Isolated Mode Operations

Operation	Syntax	Notes
Creating a keystore	`ADMINISTER KEY MANAGEMENT CREATE KEYSTORE IDENTIFIED BY *isolated_PDB_keystore_password*;`	You can create password-protected, local auto-login, and auto-login keystores in an isolated mode PDB.
Creating an auto-login keystore	`ADMINISTER KEY MANAGEMENT CREATE [LOCAL] AUTO_LOGIN KEYSTORE FROM KEYSTORE IDENTIFIED BY *isolated_PDB_keystore_password*;`	The isolated PDB knows the keystore location of both password protected and (local) auto-open wallet from *WALLET_ROOT/pdb_guid*/tde.
Opening a keystore	`ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN [FORCE KEYSTORE] IDENTIFIED BY EXTERNAL STORE | *isolated_PDB_keystore_password*;`	The `EXTERNAL_STORE` clause retrieves the isolated mode PDB keystore password from a wallet in the *tde_seps* directory in the *WALLET_ROOT*/*PDB-GUID* location.
Changing a keystore password	`ADMINISTER KEY MANAGEMENT ALTER KEYSTORE PASSWORD IDENTIFIED BY *old_isolated_PDB_keystore_password* SET *new_isolated_PDB_keystore_password* WITH BACKUP [USING '*backup_identifier*'];`	-
Backing up a TDE wallet	`ADMINISTER KEY MANAGEMENT BACKUP KEYSTORE [USING '*backup_identifier*'] IDENTIFIED BY EXTERNAL STORE | *TDE_wallet_password*] [TO '*TDE_wallet_backup_location*'] ;`	-

Table 4-1 (Cont.) ADMINISTER KEY MANAGEMENT Isolated Mode Operations

Operation	Syntax	Notes
Merging the contents of one keystore into an existing keystore	`ADMINISTER KEY MANAGEMENT MERGE KEYSTORE '*keystore1_location*' [IDENTIFIED BY *TDE_wallet1_password*] INTO EXISTING KEYSTORE '*keystore2_location*' IDENTIFIED BY *TDE_wallet2_password* WITH BACKUP [USING '*backup_identifier*'];`	-
Merging the contents of two keystores to create a third keystore	`ADMINISTER KEY MANAGEMENT MERGE KEYSTORE '*keystore1_location*' [IDENTIFIED BY *TDE_wallet1_password*] AND KEYSTORE '*keystore2_password*' [IDENTIFIED BY *TDE_wallet2_password*] INTO NEW KEYSTORE '*keystore3_location*' IDENTIFIED BY *TDE_wallet3_password*;`	-
Closing a keystore	`ADMINISTER KEY MANAGEMENT SET KEYSTORE CLOSE [IDENTIFIED BY [EXTERNAL STORE | *keystore_password*]];`	-
Closing the keystore of the CDB root when a PDB in isolated mode has its keystore open	`ADMINISTER KEY MANAGEMENT FORCE KEYSTORE CLOSE [IDENTIFIED BY [EXTERNAL STORE | *keystore_password*]];`	The `FORCE` clause allows the keystore to be closed in the CDB root even when a PDB in isolated mode still has its keystore open
Creating and activating a new TDE master encryption key (rekeying)	`ADMINISTER KEY MANAGEMENT SET [ENCRYPTION] KEY [USING TAG '*tag_name*'] [FORCE KEYSTORE] IDENTIFIED BY [EXTERNAL STORE | *keystore_password* ]WITH BACKUP [USING '*backup_identifier*'];`	-

Table 4-1 (Cont.) ADMINISTER KEY MANAGEMENT Isolated Mode Operations

Operation	Syntax	Notes
Creating a user-defined TDE master encryption key for either now (SET) or later on (CREATE)	ADMINISTER KEY MANAGEMENT SET | CREATE [ENCRYPTION] KEY '*mkid*:*mk* | *mk*' [USING ALGORITHM '*algorithm*'] [FORCE KEYSTORE] [USING TAG '*tag_name*'] IDENTIFIED BY EXTERNAL STORE | *TDE_wallet_password* WITH BACKUP [USING '*backup_identifier*'] [CONTAINER = CURRENT];	-
Activating an existing TDE master encryption key	ADMINISTER KEY MANAGEMENT USE [ENCRYPTION] KEY '*key_id*' [USING TAG 'tag'] IDENTIFIED BY [EXTERNAL STORE | *keystore_password*] WITH BACKUP [USING '*backup_identifier*'];	-
Tagging a TDE master encryption key	ADMINISTER KEY MANAGEMENT SET TAG 'tag' FOR '*key_id*' IDENTIFIED BY [EXTERNAL STORE | *keystore_password*] WITH BACKUP [USING '*backup_identifier*'];	-
Exporting a TDE master encryption key	ADMINISTER KEY MANAGEMENT EXPORT [ENCRYPTION] KEYS WITH SECRET *secret* TO '*filename*' IDENTIFIED BY *TDE_wallet_password* [WITH IDENTIFIER IN { '*key_id*' [, '*key_id*']... | (*subquery*) }];	-
Importing a TDE master encryption key	ADMINISTER KEY MANAGEMENT IMPORT [ENCRYPTION] KEYS WITH SECRET *secret* FROM '*filename*' IDENTIFIED BY *TDE_wallet_password* WITH BACKUP [USING '*backup_identifier*'];	-

Table 4-1 (Cont.) ADMINISTER KEY MANAGEMENT Isolated Mode Operations

Operation	Syntax	Notes
Migrating an encrypted database from a TDE wallet to Oracle Key Vault	`ADMINISTER KEY MANAGEMENT SET [ENCRYPTION] KEY IDENTIFIED BY *Oracle_Key_Vault_password* [FORCE KEYSTORE] MIGRATE USING *TDE_wallet_password*;`	-
Reverse-migrating an encrypted database from an external keystore to a TDE wallet	`ADMINISTER KEY MANAGEMENT SET [ENCRYPTION] KEY IDENTIFIED BY *TDE_wallet_password* REVERSE MIGRATE USING *Oracle_Key_Vault_password*;`	-
Adding a client secret	`ADMINISTER KEY MANAGEMENT ADD SECRET *'secret'* FOR CLIENT '*client_identifier*' [USING TAG '*tag_name*' ] IDENTIFIED BY [EXTERNAL STORE | *keystore_password*] WITH BACKUP [USING '*backup_identifier*'];`	-
Updating a client secret	`ADMINISTER KEY MANAGEMENT UPDATE SECRET *'secret'* FOR CLIENT '*client_identifier*' [USING TAG '*tag_name*' ] IDENTIFIED BY [EXTERNAL STORE | *keystore_password*] WITH BACKUP [USING '*backup_identifier*'];`	-
Deleting a client secret	`ADMINISTER KEY MANAGEMENT DELETE SECRET FOR CLIENT '*client_identifier* IDENTIFIED BY [EXTERNAL STORE | *keystore_password*] WITH BACKUP [USING '*backup_identifier*'];`	-

Table 4-1 (Cont.) ADMINISTER KEY MANAGEMENT Isolated Mode Operations

Operation	Syntax	Notes
Isolating a PDB	`ADMINISTER KEY MANAGEMENT ISOLATE KEYSTORE IDENTIFIED BY *isolated_keystore_password* FROM ROOT KEYSTORE [FORCE KEYSTORE] IDENTIFIED BY [EXTERNAL STORE | *united_keystore_password*] WITH BACKUP [USING *backup_id*];`	This operation performs two actions. First, it changes the `TDE_CONFIGURATION` of the PDB so that it is in isolated mode. Second, it moves the PDB's current and previously active TDE master encryption keys from the root keystore to a newly-created isolated keystore of the PDB, having its own isolated keystore password, where the PDB will be able to manage its own keys.

4.3 Operations That Are Not Allowed in an Isolated Mode PDB

There are several `ADMINISTER KEY MANAGEMENT` operations that you cannot perform in an isolated mode PDB.

These operations include the following:

- Using the `CONTAINER = ALL` clause to create a new TDE master encryption key for later use in each pluggable database (PDB)
- Moving encryption keys from the keystore of the CDB root into a keystore of a PDB that is configured in isolated mode

4.4 Configuring the Keystore Location and Type for Isolated Mode

For isolated mode, you can configure the keystore location and type by using only parameters or a combination of parameters and the `ALTER SYSTEM` statement.

4.4.1 About Configuring the Keystore Location and Type for Isolated Mode

Configuring the `WALLET_ROOT` and `TDE_CONFIGURATION` parameters for the CDB environment is similar to the procedure used for united mode.

The difference is that rather than using the `RESET` clause of the `ALTER SYSTEM` statement, you use the `SET` clause. You can perform the configuration by adding the `WALLET_ROOT` and `TDE_CONFIGURATION` parameters to the initialization parameter file. To configure a PDB in isolated mode, you must set a value for the `TDE_CONFIGURATION` parameter of the PDB, which you can do either by using the `ALTER SYSTEM` statement or by issuing the `ADMINISTER KEY MANAGEMENT ISOLATE KEYSTORE` statement. This section focuses on the use of the `ALTER SYSTEM` statement.

Depending on whether your system uses pfile or spfile, you must set the `SCOPE` clause in the `ALTER SYSTEM` statement appropriately when setting the value of the `TDE_CONFIGURATION` parameter for the PDB. The value of the `TDE_CONFIGURATION` parameter is a list of attribute-

value pairs, and it is the value of the KEYSTORE_CONFIGURATION attribute that specifies the type of the keystore, as follows:

- FILE specifies a TDE wallet.
- OKV specifies Oracle Key Vault.
- FILE|OKV specifies a reverse-migration from the OKV keystore type to the FILE keystore type has occurred.
- FILE|HSM specifies a reverse-migration from the HSM keystore type to the FILE keystore type has occurred.
- OKV|FILE specifies a migration from the FILE keystore type to the OKV keystore type has occurred. The keystore type has two meanings: it either means that you are migrating from FILE to OKV, or it means that the configuration started out as using OKV but is now using an auto-login OKV configuration, where the OKV password resides in a cwallet.sso file in the *WALLET_ROOT*/*pdb_guid*/tde directory.

After you have used ALTER SYSTEM to configure the TDE_CONFIGURATION value for the selected PDB, the PDB in the CDB environment is in isolated mode. The steps in this procedure explain in detail how to configure an individual PDB to be in isolated mode, using its own keystore type.

4.4.2 Configuring the Keystore Location and Keystore Type for an Isolated Mode PDB

You can configure isolated mode by setting WALLET_ROOT in the initialization parameter file in the CDB root and TDE_CONFIGURATION in the PDB you want to isolate.

1. In the root, ensure that the WALLET_ROOT parameter is set correctly.

   For example:

   ```
   SHOW PARAMETER WALLET_ROOT
   ```

2. As a user who has the ADMINISTER KEY MANAGEMENT or SYSKM privilege, run the following statement in the PDB:

   ```
   ADMINISTER KEY MANAGEMENT CREATE KEYSTORE IDENTIFIED BY
   "iso_PDB_TDE_wallet_password";
   ```

   This statement does the following:

   - Sets the TDE_CONFIGURATION in the isolated PDB to FILE.
   - Creates the directories *pdb_guid*/tde under WALLET_ROOT.
   - Creates a new TDE wallet with its own password (which might be not known to the CDB administrator).

3. Check the configuration.

   - To check the TDE_CONFIGURATION parameter setting in the isolated PDB:

     ```
     SHOW PARAMETER TDE_CONFIGURATION
     ```

     The output should reflect the keystore configuration that you set for the current PDB. If it shows a different keystore configuration (for example, FILE if you had set it to OKV), then the setting may be showing the keystore configuration that was set for the CDB root, in united mode.

- To check the keystore mode:

  ```
  SELECT KEYSTORE_MODE FROM V$ENCRYPTION_WALLET;
  ```

  The output should be `ISOLATED`.

After you configure isolated mode, the CDB root keystore that was available to the PDB when it was in united mode is no longer available to this PDB. At this stage, the PDB is configured to use its own keystore. If the `KEYSTORE_CONFIGURATION` parameter was `FILE` (meaning that the PDB is configured to use a TDE wallet), then the keystore location configured for the PDB is `WALLET_ROOT/PDB-GUID/tde`. If a keystore exists at that location and contains a TDE master encryption key, then that key is only available to this PDB, not to any other PDB. If no keystore exists at that location, you now can now proceed to create a TDE wallet and set a TDE master encryption key. If you later decide that you want the isolated mode PDB to become a united mode PDB again, then you can use the `ADMINISTER KEY MANAGEMENT UNITE KEYSTORE` statement. When you run `ADMINISTER KEY MANAGEMENT UNITE KEYSTORE`, it moves the keys from the PDB's keystore to the keystore of the CDB root, but it leaves any client secrets behind. So if there were no client secrets in the first place, then it would leave the PDB's keystore essentially "empty". It can now be backed up, and removed. Always back up keystores before you remove them, even empty keystores.

Related Topics

- Configuring United Mode with the Initialization Parameter File and ALTER SYSTEM
  If your environment relies on server parameter files (`spfile`), then you can set `WALLET_ROOT` and `TDE_CONFIGURATION` using `ALTER SYSTEM SET` with `SCOPE`.
- Uniting a Pluggable Database Keystore
  Uniting a PDB keystore moves the TDE master encryption keys from the PDB keystore into the keystore of the CDB root. This enables the administrator of the keystore of the CDB root to manage the keys.
- Example: Configuring a TDE Wallet When Multiple Databases Share the Same Host
  You can configure multiple databases to share the same host by setting the `WALLET_ROOT` parameter.
- Example: Configuring a TDE Wallet for an Oracle Automatic Storage Management Disk Group
  In an Oracle Real Applications Clusters (Oracle RAC) environment, the `WALLET_ROOT` parameter points to a shared directory in Oracle Automatic Storage Management (ASM) that is accessible from all Oracle RAC instances of that database.

4.4.3 Example: Restoring an Older Version of a Control File

You can set `TDE_CONFIGURATION` if you have an older version of a control file that must be restored and only a few PDBs were configured in isolated mode.

When the CDB root and the PDB are both in the mount state, then you can only change the PDB’s keystore configuration from the CDB root.

1. Log in to the CDB root as a user who was granted the `SYSDBA` administrative privilege.
2. For each PDB that you want to change, use the following syntax:

   ```
   ALTER SYSTEM SET
   TDE_CONFIGURATION="CONTAINER=pdb_name;KEYSTORE_CONFIGURATION=keystore_type"
   SCOPE=memory;
   ```

   For example, for the `hrpdb` and `salespdb` PDBs using `FILE` (for TDE wallets) as the keystore type:

```
ALTER SYSTEM SET TDE_CONFIGURATION="CONTAINER=hrpdb;KEYSTORE_CONFIGURATION=FILE"
SCOPE=memory;
ALTER SYSTEM SET TDE_CONFIGURATION="CONTAINER=salespdb;KEYSTORE_CONFIGURATION=FILE"
SCOPE=memory;
```

3. After you set the `TDE_CONFIGURATION` parameter for each PDB, log in to the CDB root and then set `TDE_CONFIGURATION` for the CDB root itself.

   ```
   ALTER SYSTEM SET TDE_CONFIGURATION="KEYSTORE_CONFIGURATION=FILE";
   ```

   At this stage, CDB root is in the mounted state. The value of the `TDE_CONFIGURATION` parameter that was set using `ALTER SYSTEM` with the `CONTAINER` attribute is only present in the memory of the CDB root. To ensure that the configuration is properly applied to each PDB, you must close and then reopen the PDB. When an isolated mode PDB is opened, the configuration set by the `ALTER SYSTEM` statement that was issued in the CDB root is read from the control file and then is automatically applied to the PDB.

4. Connect to each PDB and then close and reopen the PDB.

   ```
   ALTER PLUGGABLE DATABASE pdb_name CLOSE IMMEDIATE;
   ALTER PLUGGABLE DATABASE pdb_name OPEN;
   ```

4.4.4 Example: Addressing the Problem of a Lost Control File

You can address the problem of a lost control file by using the `ALTER SYSTEM` statement.

Running these statements with `SCOPE` set to memory will store the `CONTAINER` value in memory. When you open the isolated PDB, this configuration will automatically be updated for the PDB.

If you are using an Oracle Data Guard environment, then to correct the control file, run these statements on both the primary and the standby databases.

1. Log in to the CDB root as a user who was granted the `SYSDBA` administrative privilege.
2. If you are unsure of the exact state of the system, then you should run `ALTER SYSTEM` with `RESET`.

   For example:

   ```
   ALTER SYSTEM RESET TDE_CONFIGURATION SCOPE=memory;
   ```

3. For each PDB that you want to change, use the following syntax:

   ```
   ALTER SYSTEM SET TDE_CONFIGURATION="CONTAINER=pdb_name;KEYSTORE_CONFIGURATION=FILE"
   SCOPE=memory;
   ```

   For example, for the `hrpdb` and `salespdb` PDBs with `FILE` (for TDE wallets) as the keystore type:

   ```
   ALTER SYSTEM SET TDE_CONFIGURATION="CONTAINER=hrpdb;KEYSTORE_CONFIGURATION=FILE"
   SCOPE=memory;
   ALTER SYSTEM SET TDE_CONFIGURATION="CONTAINER=salespdb;KEYSTORE_CONFIGURATION=FILE"
   SCOPE=memory;
   ```

4. After you set the `TDE_CONFIGURATION` parameter for each PDB, log in to the CDB root and then set `TDE_CONFIGURATION` for the CDB root itself.

   ```
   ALTER SYSTEM SET TDE_CONFIGURATION="KEYSTORE_CONFIGURATION=FILE";
   ```

   At this stage, CDB root is in the mounted state. The value of the `TDE_CONFIGURATION` parameter that was set using `ALTER SYSTEM` with the `CONTAINER` attribute is only present in the memory of the CDB root. To ensure that the configuration is properly applied to each PDB, you must close and then reopen the PDB. When an isolated mode PDB is opened,

the configuration set by the `ALTER SYSTEM` statement that was issued in the CDB root is read from the control file and then is automatically applied to the PDB.

5. Connect to each PDB and then close and reopen the PDB.

```
ALTER PLUGGABLE DATABASE pdb_name CLOSE IMMEDIATE;
ALTER PLUGGABLE DATABASE pdb_name OPEN;
```

4.4.5 Example: Configuring Isolated Mode in an Oracle Real Application Clusters Environment

You can use `ALTER SYSTEM` to configure isolated mode in an Oracle Real Application Clusters (Oracle RAC) environment.

- To ensure that the effect of the `ALTER SYSTEM` statement is applied on each Oracle RAC node, specify the wildcard (`*`) in the `SID` clause of the `ALTER SYSTEM` statement, as follows. You can run this statement from either the CDB root or a PDB.

```
ALTER SYSTEM SET TDE_CONFIGURATION="KEYSTORE_CONFIGURATION=keystore_type" SID='*';
```

4.5 Configuring a TDE Wallet and TDE Master Encryption Key in Isolated Mode

In isolated mode, the TDE wallet is associated with a PDB.

4.5.1 About Configuring a TDE Wallet in Isolated Mode

You can create all types of TDE wallets in isolated mode: password-protected, password protected with the credential provided from an external store, auto-login, local auto-login.

To enable encryption in the PDB after it is configured in isolated mode with the `KEYSTORE_CONFIGURATION` attribute set to `FILE` (that is, to use a TDE wallet), you must create a TDE wallet, open the TDE wallet, and then set a TDE master encryption key in the TDE wallet. Afterward, you can begin to encrypt data for tables and tablespaces that will be accessible in the PDB.

In a multitenant environment, you can create a secure external store to hold the credentials of the TDE wallet. This feature enables you to hide the keystore password: it removes the need for storing the wallet password in any script or tool that accesses the database without user intervention, such as an overnight batch script. When the `WALLET_ROOT` parameter is specified, the location of the external store for the CDB root is *`WALLET_ROOT`*`/tde_seps` and for the PDB it is *`WALLET_ROOT`*`/`*`pdb_guid`*`/tde_seps`. When the `WALLET_ROOT` parameter is set, there is no longer a single central external store, so when a keystore password is updated, the corresponding external store must be updated as well. When the `WALLET_ROOT` parameter is not specified, then the location of the external store is the same for both the CDB root and for every PDB. The external store location must then be set by the `EXTERNAL_KEYSTORE_CREDENTIAL_LOCATION` initialization parameter. When the `WALLET_ROOT` parameter is not specified, then there is a single central external store, so when you update the keystore password, only the central external store at the `EXTERNAL_KEYSTORE_CREDENTIAL_LOCATION` must be updated.

In a multitenant environment, different PDBs can access this external store location when you run the `ADMINISTER KEY MANAGEMENT` statement using the `IDENTIFIED BY EXTERNAL STORE` clause. This way, you can centrally locate the password and then update it only once in the external store.

Related Topics

- Configuring an External Store for a Keystore Password
  An external store for a keystore password stores the keystore password in a centrally accessed and managed location.

4.5.2 Step 1: Create a TDE Wallet in a PDB Configured in Isolated Mode

A password-protected TDE wallet requires a password to protect the keystore keys and credentials.

1. Connect to the isolated mode PDB as a user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.
2. Use the `SHOW PARAMETER` command to confirm that `WALLET_ROOT` is set, and `TDE_CONFIGURATION` is set to `KEYSTORE_CONFIGURATION=FILE`.
3. Run the `ADMINISTER KEY MANAGEMENT` SQL statement to create the TDE wallet using the following syntax:

   ```
   ADMINISTER KEY MANAGEMENT CREATE KEYSTORE IDENTIFIED BY TDE_wallet_password;
   ```

   This command creates the `/tde` directory under `WALLET_ROOT/pdb_guid` (unless it already exists), and creates a password-protected TDE wallet in this directory. If `WALLET_ROOT` points to a shared directory in ASM (`+DATA/DB_UNIQUE_NAME`), then the `pdb_guid/tde` subdirectory and the TDE wallet is auto-created by the `CREATE KEYSTORE` statement.

   In this specification, *`TDE_wallet_password`* is the password of the TDE wallet that you, the security administrator, creates.

   For example, to create the TDE wallet in the `WALLET_ROOT/pdb_guid/tde` directory:

   ```
   ADMINISTER KEY MANAGEMENT CREATE KEYSTORE IDENTIFIED BY password;

   keystore altered.
   ```

After you complete these steps, the `ewallet.p12` file, which is the TDE wallet, appears in the TDE wallet location.

Related Topics

- Creating an Auto-Login or a Local Auto-Login TDE Wallet
  As an alternative to password-protected TDE wallets, you can create either an auto-login or local auto-login TDE wallet.

4.5.3 Step 2: Open the TDE Wallet in an Isolated Mode PDB

To open a TDE wallet in isolated mode, you must use the `ADMINISTER KEY MANAGEMENT` statement with the `SET KEYSTORE OPEN` clause.

1. Connect to the isolated mode PDB as a user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.
2. Run the `ADMINISTER KEY MANAGEMENT` statement to open the TDE wallet.

   ```
   ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN
   IDENTIFIED BY TDE_wallet_password | EXTERNAL STORE;

   Keystore altered.
   ```

To switch over to opening the password-protected TDE wallet when an auto-login wallet is configured and is currently open, specify the `FORCE KEYSTORE` clause as follows.

```
ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN
FORCE KEYSTORE
IDENTIFIED BY TDE_wallet_password | EXTERNAL STORE;

Keystore altered.
```

Here, the `IDENTIFIED BY EXTERNAL STORE` clause is included in the statement because the wallet credentials exist in an external store. This enables the password-protected TDE wallet to be opened without specifying the TDE wallet password within the statement itself.

If the `WALLET_ROOT` parameter has been set, then Oracle Database finds the external store by searching in this path: *`WALLET_ROOT`*`/`*`pdb_guid`*`/tde_seps`.

3. Confirm that the TDE wallet is open.

```
SELECT STATUS FROM V$ENCRYPTION_WALLET;
```

Related Topics

- About Opening TDE Wallets
  A password-protected TDE wallet must be open before any TDE master encryption keys can be created or accessed in the wallet.

4.5.4 Step 3: Set the TDE Master Encryption Key in the TDE Wallet of the Isolated Mode PDB

To set the TDE master encryption key in a TDE wallet in an isolated mode PDB, use the `ADMINISTER KEY MANAGEMENT` statement with the `SET KEY` clause.

1. Connect to the isolated mode PDB as a user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.
2. Ensure that the database is open in `READ WRITE` mode.

   To find the status, run the `show pdbs` command.
3. Run the `ADMINISTER KEY MANAGEMENT` SQL statement to set the key in the TDE wallet.

   For example, if the TDE wallet of the PDB is password-protected, the PDB is open, and the TDE wallet of the PDB is open:

```
ADMINISTER KEY MANAGEMENT SET KEY
IDENTIFIED BY TDE_wallet_password
WITH BACKUP USING 'emp_key_backup';

keystore altered.
```

   If the TDE wallet is closed:

```
ADMINISTER KEY MANAGEMENT SET KEY
FORCE KEYSTORE
IDENTIFIED BY TDE_wallet_password
WITH BACKUP USING 'emp_key_backup';

keystore altered.
```

   In this specification:

- `FORCE KEYSTORE` should be included if the TDE wallet is closed. This automatically opens the TDE wallet before setting the TDE master encryption key. The `FORCE KEYSTORE` clause also switches over to opening the password-protected TDE wallet when an auto-login TDE wallet is configured and is currently open.
- `IDENTIFIED BY` specifies the TDE wallet password. Alternatively, if the TDE wallet password is in an external store, you can use the `IDENTIFIED BY EXTERNAL STORE` clause.

4. Confirm that the TDE master encryption key is set.

```
SELECT MASTERKEY_ACTIVATED FROM V$DATABASE_KEY_INFO;
```

The output should be `YES`.

Related Topics

- About Setting the TDE Wallet TDE Master Encryption Key
  The TDE master encryption key is stored in the TDE wallet.

4.5.5 Step 4: Encrypt Your Data in Isolated Mode

Now that you have completed the configuration, you can begin to encrypt data in the PDB.

Related Topics

- Encrypting Columns in Tables
  You can use Transparent Data Encryption to encrypt individual columns in database tables.
- Encryption Conversions for Tablespaces and Databases
  You can perform encryption operations on both offline and online tablespaces and databases.

4.6 Configuring a Container Database with Isolated Mode PDBs for Oracle Key Vault

Isolated PDBs have their own virtual wallet, with their own password, in Oracle Key Vault.

4.6.1 About Configuring an External Keystore in Isolated Mode

You can configure an external keystore for a PDB when the PDB is configured in isolated mode.

To configure an external keystore for a PDB in isolated mode, you first must set the `WALLET_ROOT` parameter. This is necessary for two reasons: first, to have support for migrating to a TDE wallet in the future, and second, because the configuration file for Oracle Key Vault is retrieved from a location under `WALLET_ROOT`. Afterwards, you must set the `KEYSTORE_CONFIGURATION` attribute of the `TDE_CONFIGURATION` parameter to `OKV`, open the configured external keystore, and then set the TDE master encryption key for the PDB. After you complete these tasks, you can begin to encrypt data in your database.

How you specify the `IDENTIFIED BY` clause when you run the `ADMINISTER KEY MANAGEMENT` statement depends on the type of external keystore. Use the following syntax:

```
IDENTIFIED BY EXTERNAL STORE|Oracle_Key_Vault_password
```

Enter the Oracle Key Vault password that was given during the Oracle Key Vault client installation. If at that time no password was given, then the password in the `ADMINISTER KEY MANAGEMENT` statement becomes `NULL`.

4.6.2 Step 1: Configure Isolated PDBs for Oracle Key Vault

You can configure isolated mode PDBs for Oracle Key Vault by setting the `TDE_CONFIGURATION` parameter.

1. If the `WALLET_ROOT` parameter is set, then install the Oracle Key Vault client software into the `WALLET_ROOT/pdb_guid/okv` directory.
2. Log in to the database instance as a user who has been granted the `ALTER SYSTEM` administrative privilege.
3. Set the `TDE_CONFIGURATION` dynamic initialization parameter to specify the keystore type by using the following syntax:

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

   In this specification:

   - *keystore_type* can be `OKV`, to configure a password-protected Oracle Key Vault keystore, or `OKV|TDE` for an auto-open connection into Oracle Key Vault.

   For example, to configure your database to use Oracle Key Vault:

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

4.6.3 Step 2: Open the Isolated Mode PDB External Keystore

If the isolated PDB does not have an auto-open connection into the external keystore, then you must open it manually before your open the PDB.

1. Connect to the isolated mode PDB as a user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.
2. Open the external keystore as follows, by enclosing the password in double quotation marks:

   For example:

   ```
   ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN
   IDENTIFIED BY "Oracle_Key_Vault_password" | EXTERNAL STORE;
   ```

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

   You must open the keystore of the CDB root first.

Related Topics

- About Opening the Connection to Oracle Key Vault
  You must open the connection to Oracle Key Vault so that it is accessible to the database before you can perform any encryption or decryption.

4.6.4 Step 3: Set the First TDE Master Encryption Key in the External Keystore

After you have opened the external keystore in an isolated mode PDB, you are ready to set the TDE master encryption key for the PDB.

4.6.4.1 Setting the TDE Master Encryption Key in the Isolated Mode External Keystore

You should complete this procedure if you have not previously configured an external keystore for Transparent Data Encryption.

1. Connect to the isolated mode PDB as a user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.
2. Ensure that the database is open in `READ WRITE` mode.

   You can set the TDE master encryption key if `OPEN_MODE` is set to `READ WRITE`. To find the status, run the `show pdbs` command.
3. To configure the heartbeat batch size, set the `HEARTBEAT_BATCH_SIZE` initialization parameter.

   The `HEARTBEAT_BATCH_SIZE` parameter configures the size of the "batch of heartbeats" sent per heartbeat period to the external key manager. Enter a value between 2 and 100. The default value is 5. The default duration of the heartbeat period is three seconds. By setting the heartbeat batch size, you can stagger the heartbeats across batches of PDBs to ensure that for each batch a heartbeat can be completed for each PDB within the batch during the heartbeat period, and also ensure that the TDE master encryption key of the PDB can be reliably fetched from an Oracle Key Vault server and cached in the Oracle Key Vault persistent cache. (See Heartbeat Batch Size for External Keystores for details about how `HEARTBEAT_BATCH_SIZE` works.)

   ```
   ALTER SYSTEM SET HEARTBEAT_BATCH_SIZE=30 SCOPE=BOTH SID='*';
   ```
4. Set the new TDE master encryption key by using the following syntax:

   ```
   ADMINISTER KEY MANAGEMENT SET KEY
   [USING TAG 'tag']
   [FORCE KEYSTORE]
   IDENTIFIED BY EXTERNAL STORE|Oracle_Key_Vault_password;
   ```

   In this specification:

   - `FORCE KEYSTORE` temporarily opens the password-protected TDE wallet for this operation if the TDE wallet is closed, if an auto-login TDE wallet is configured and is currently open, or if a password-protected TDE wallet is configured and is currently closed.
   - `IDENTIFIED BY` can be one of the following settings:
     - `EXTERNAL STORE` uses the keystore password stored in the external store to perform the keystore operation.
     - *`Oracle_Key_Vault_password`* is the password that was given during the Oracle Key Vault client installation. If at that time no password was given, then the password in the `ADMINISTER KEY MANAGEMENT` statement becomes `NULL`.

   For example:

   ```
   ADMINISTER KEY MANAGEMENT SET KEY
   USING TAG 'sessionid=3205062574:terminal=xcvt'
   FORCE KEYSTORE
   IDENTIFIED BY EXTERNAL STORE;

   keystore altered.
   ```

5. Confirm that the TDE master encryption key is set.

```
SELECT MASTERKEY_ACTIVATED FROM V$DATABASE_KEY_INFO;
```

The output should be `YES`.

Related Topics

- About Setting the External Keystore TDE Master Encryption Key
  You must create a TDE master encryption key that is stored inside the external keystore.

4.6.4.2 Migration of a Previously Configured Encryption Key in Isolated Mode

You must migrate the previously configured master encryption key if you previously configured a TDE wallet.

Related Topics

- Migration of an Encrypted Database from a TDE Wallet to Oracle Key Vault or OCI KMS
  To switch from a TDE wallet to centralized key management with Oracle Key Vault or Oracle Cloud Infrastructure (OCI) Key Management Service (KMS), after you upload all current and retired TDE master keys you must migrate the database from the TDE wallet to Oracle Key Vault or OCI KMS.

4.6.5 Step 4: Encrypt Your Data in Isolated Mode

Now that you have completed the keystore configuration and the PDB is configured in isolated mode, you can begin to encrypt data in the PDB.

Related Topics

- Encrypting Columns in Tables
  You can use Transparent Data Encryption to encrypt individual columns in database tables.
- Encryption Conversions for Tablespaces and Databases
  You can perform encryption operations on both offline and online tablespaces and databases.
- *Oracle Key Vault Administrator's Guide*

5

Encrypting Columns in Tables

You can use Transparent Data Encryption to encrypt individual columns in database tables.

5.1 About Encrypting Columns in Tables

You can encrypt individual columns in tables.

Whether you choose to encrypt individual columns or entire tablespaces depends on the data types that the table has. There are also several features that do not support TDE column encryption.

Related Topics

- Data Types That Can Be Encrypted with TDE Column Encryption
  Oracle Database supports a specific set of data types that can be used with TDE column encryption.
- Restrictions on Using TDE Column Encryption
  TDE column encryption is performed at the SQL layer. Oracle Database utilities that bypass the SQL layer cannot use TDE column encryption services.

5.2 Data Types That Can Be Encrypted with TDE Column Encryption

Oracle Database supports a specific set of data types that can be used with TDE column encryption.

You can encrypt data columns that use a variety of different data types.

Supported data types are as follows:

- `BINARY_DOUBLE`
- `BINARY_FLOAT`
- `CHAR`
- `DATE`
- `INTERVAL DAY TO SECOND`
- `INTERVAL YEAR TO MONTH`
- `NCHAR`
- `NUMBER`
- `NVARCHAR2`
- `RAW` (legacy or extended)
- `TIMESTAMP` (includes `TIMESTAMP WITH TIME ZONE` and `TIMESTAMP WITH LOCAL TIME ZONE`)
- `VARCHAR2` (legacy or extended)

If you want to encrypt large binary objects (LOBs), then you can use Oracle SecureFiles. Oracle SecureFiles enables you to store LOB data securely. To encrypt a LOB using SecureFiles, you use the `CREATE TABLE` or `ALTER TABLE` statements.

You cannot encrypt a column if the encrypted column size is greater than the size allowed by the data type of the column.

Table 5-1 shows the maximum allowable sizes for various data types.

Table 5-1 Maximum Allowable Size for Data Types

Data Type	Maximum Size
`CHAR`	1932 bytes
`VARCHAR2` (legacy)	3932 bytes
`VARCHAR2` (extended)	32,699 bytes
`NVARCHAR2` (legacy)	1966 bytes
`NVARCHAR2` (extended)	16,315 bytes
`NCHAR`	966 bytes
`RAW` (extended)	32,699 bytes

> **Note:**
>
> TDE tablespace encryption does not have these data type restrictions.

Related Topics

- Restrictions on Using Transparent Data Encryption Tablespace Encryption
  You should be aware of restrictions on using Transparent Data Encryption when you encrypt a tablespace.
- *Oracle Database SecureFiles and Large Objects Developer's Guide*

5.3 Restrictions on Using TDE Column Encryption

TDE column encryption is performed at the SQL layer. Oracle Database utilities that bypass the SQL layer cannot use TDE column encryption services.

Do not use TDE column encryption with the following database features:

- Index types other than B-tree
- Range scan search through an index
- Synchronous change data capture
- Transportable tablespaces
- Columns that have been created as identity columns

In addition, you cannot use TDE column encryption to encrypt columns used in foreign key constraints.

Applications that must use these unsupported features can use the `DBMS_CRYPTO` PL/SQL package for their encryption needs.

Transparent Data Encryption protects data stored on a disk or other media. It does not protect data in transit. Use the network encryption solutions discussed in *Oracle Database Security Guide* to encrypt data over the network.

Related Topics

- How Transparent Data Encryption Works with Export and Import Operations
  Oracle Data Pump can export and import tables that contain encrypted columns, as well as encrypt entire dump sets.
- Data Types That Can Be Encrypted with TDE Column Encryption
  Oracle Database supports a specific set of data types that can be used with TDE column encryption.
- *Oracle Database Security Guide*

5.4 Creating Tables with Encrypted Columns

Oracle Database provides a selection of different algorithms that you can use to define the encryption used in encrypted columns.

5.4.1 About Creating Tables with Encrypted Columns

You can use the `CREATE TABLE` SQL statement to create a table with an encrypted column.

To create relational tables with encrypted columns, you can specify the SQL `ENCRYPT` clause when you define database columns with the `CREATE TABLE` SQL statement.

5.4.2 Creating a Table with an Encrypted Column Using the Default Algorithm

By default, TDE uses the `AES` encryption algorithm with a 256-bit key length (`AES256`).

If you encrypt a table column without specifying an algorithm, then the column is encrypted using the `AES256` algorithm. TDE adds salt to plaintext before encrypting it. Adding salt makes it harder for attackers to steal data through a brute force attack. TDE also adds a Message Authentication Code (MAC) to the data for integrity checking. The `SHA-1` integrity algorithm is used by default. (Starting with Oracle Database release 21c, `SHA-1` is deprecated. If you use TDE column encryption, then Oracle recommends that you implement TDE tablespace encryption instead.)

- To create a table that encrypts a column, use the `CREATE TABLE` SQL statement with the `ENCRYPT` clause.

  For example, to encrypt a table column using the default algorithm:

  ```
  CREATE TABLE employee (
       first_name VARCHAR2(128),
       last_name VARCHAR2(128),
       empID NUMBER,
       salary NUMBER(6) ENCRYPT);
  ```

  This example creates a new table with an encrypted column (`salary`). The column is encrypted using the default encryption algorithm (`AES256`). Salt and MAC are added by default. This example assumes that the keystore is open and a master encryption key is set.

> **Note:**
>
> If there are multiple encrypted columns in a table, then all of these columns must use the same pair of encryption and integrity algorithms.
>
> Salt is specified at the column level. This means that an encrypted column in a table can choose not to use salt irrespective of whether or not other encrypted columns in the table use salt.

5.4.3 Creating a Table with an Encrypted Column Using No Algorithm or a Non-Default Algorithm

You an use the `CREATE TABLE` SQL statement to create a table with an encrypted column.

By default, TDE adds salt to plaintext before encrypting it. Adding salt makes it harder for attackers to steal data through a brute force attack. However, if you plan to index the encrypted column, then you must use the `NO SALT` parameter.

- To create a table that uses an encrypted column that is a non-default algorithm or no algorithm, run the `CREATE TABLE` SQL statement as follows:
  - If you do not want to use any algorithm, then include the `ENCRYPT NO SALT` clause.
  - If you want to use a non-default algorithm, then use the `ENCRYPT USING` clause, followed by one of the following algorithms enclosed in single quotation marks:
    - `3DES168`
    - `AES128`
    - `AES192` (default)
    - `AES256` (default)

    The following example shows how to specify encryption settings for the `empID` and `salary` columns.

    ```
    CREATE TABLE employee (
         first_name VARCHAR2(128),
         last_name VARCHAR2(128),
         empID NUMBER ENCRYPT NO SALT,
         salary NUMBER(6) ENCRYPT USING 'AES128');
    ```

    In this example:

    - The `empID` column is encrypted and does not use salt. Both the `empID` and `salary` columns will use the `AES128` encryption algorithm, because all of the encrypted columns in a table must use the same encryption algorithm.
    - The `salary` column is encrypted using the `AES128` encryption algorithm. Note that the string that specifies the algorithm must be enclosed in single quotation marks (' '). The `salary` column uses salt by default.

5.4.4 Using the NOMAC Parameter to Save Disk Space and Improve Performance

You can bypass checks that Transparent Data Encryption (TDE) performs. This can save up to 20 bytes of disk space per encrypted value.

If the number of rows and encrypted columns in the table is large, then bypassing TDE checks can add up to a significant amount of disk space. In addition, this saves processing cycles and reduces the performance overhead associated with TDE. TDE uses the `SHA-1` integrity algorithm by default. (Starting with Oracle Database release 21c, `SHA-1` is deprecated. If you use TDE column encryption, then Oracle recommends that you implement TDE tablespace encryption instead.) All of the encrypted columns in a table must use the same integrity algorithm. If you already have a table column using the `SHA-1` algorithm, then you cannot use the `NOMAC` parameter to encrypt another column in the same table.

- To bypass the integrity check during encryption and decryption operations, use the `NOMAC` parameter in the `CREATE TABLE` and `ALTER TABLE` statements.

Related Topics

- Performance and Storage Overhead of Transparent Data Encryption
  The performance of Transparent Data Encryption can vary.

5.4.5 Example: Using the NOMAC Parameter in a CREATE TABLE Statement

You can use the `CREATE TABLE` SQL statement to encrypt a table column using the `NOMAC` parameter.

Example 5-1 creates a table with an encrypted column. The `empID` column is encrypted using the `NOMAC` parameter.

Example 5-1 Using the NOMAC parameter in a CREATE TABLE statement

```
CREATE TABLE employee (
     first_name VARCHAR2(128),
     last_name VARCHAR2(128),
     empID NUMBER ENCRYPT 'NOMAC' ,
     salary NUMBER(6));
```

5.4.6 Example: Changing the Integrity Algorithm for a Table

You can use the `ALTER TABLE` SQL statement in different foregrounds to convert different offline tablespaces in parallel.

Example 5-2 shows how to change the integrity algorithm for encrypted columns in a table. The encryption algorithm is set to `AES256` and the integrity algorithm is set to `SHA-1`. The second `ALTER TABLE` statement sets the integrity algorithm to `NOMAC`.

Example 5-2 Changing the Integrity Algorithm for a Table

```
ALTER TABLE EMPLOYEE REKEY USING 'AES256' 'SHA-1';

ALTER TABLE EMPLOYEE REKEY USING 'AES256' 'NOMAC';
```

5.4.7 Creating an Encrypted Column in an External Table

The external table feature enables you to access data in external sources as if the data were in a database table.

External tables can be updated using the `ORACLE_DATAPUMP` access driver.

- To encrypt specific columns in an external table, use the `ENCRYPT` clause when you define those columns:

  A system-generated key encrypts the columns. For example, the following `CREATE TABLE` SQL statement encrypts the `ssn` column using the `AES192` algorithm:

```
CREATE TABLE emp_ext (
    first_name,
    ....
    ssn ENCRYPT USING 'AES192',
    ....
```

  If you plan to move an external table to a new location, then you cannot use a randomly generated key to encrypt the columns. This is because the randomly generated key will not be available at the new location.

  For such scenarios, you should specify a password while you encrypt the columns. After you move the data, you can use the same password to regenerate the key required to access the encrypted column data at the new location.

  Table partition exchange also requires a password-protected TDE table key.

  The following example creates an external table using a password to create the TDE table key:

```
CREATE TABLE emp_ext (
     first_name,
     last_name,
     empID,
     salary,
     ssn ENCRYPT IDENTIFIED BY password
)  ORGANIZATION EXTERNAL
   (
    TYPE ORACLE_DATAPUMP
    DEFAULT DIRECTORY "D_DIR"
    LOCATION('emp_ext.dat')
    )
    REJECT LIMIT UNLIMITED
AS SELECT * FROM EMPLOYEE;
```

5.5 Encrypting Columns in Existing Tables

You can encrypt columns in existing tables. As with new tables, you have a choice of different algorithms to use to definite the encryption.

5.5.1 About Encrypting Columns in Existing Tables

The `ALTER TABLE` SQL statement enables you to encrypt columns in an existing table.

To add an encrypted column to an existing table, or to encrypt or decrypt an existing column, you use the `ALTER TABLE` SQL statement with the `ADD` or `MODIFY` clause.

5.5.2 Adding an Encrypted Column to an Existing Table

You can encrypt columns in existing tables, use a different algorithm, and use `NO SALT` to index the column.

- To add an encrypted column to an existing table, use the `ALTER TABLE ADD` statement, specifying the new column with the `ENCRYPT` clause.

  The following example adds an encrypted column, `ssn`, to an existing table, called `employee`. The `ssn` column is encrypted with the default `AES256` algorithm. Salt and MAC are added by default.

  ```
  ALTER TABLE employee ADD (ssn VARCHAR2(11) ENCRYPT);
  ```

5.5.3 Encrypting an Unencrypted Column

You can use the `ALTER TABLE MODIFY` statement to encrypt an existing unencrypted column.

- To encrypt an existing unencrypted column, use the `ALTER TABLE MODIFY` statement, specifying the unencrypted column with the `ENCRYPT` clause.

  The following example encrypts the `first_name` column in the `employee` table. The `first_name` column is encrypted with the default `AES192` algorithm. Salt is added to the data, by default. You can encrypt the column using a different algorithm. If you want to index a column, then you must specify `NO SALT`. You can also bypass integrity checks by using the `NOMAC` parameter.

  ```
  ALTER TABLE employee MODIFY (first_name ENCRYPT);
  ```

  The following example encrypts the `first_name` column in the employee table using the `NOMAC` parameter.

  ```
  ALTER TABLE employee MODIFY (first_name ENCRYPT 'NOMAC');
  ```

5.5.4 Decrypting an Application Table Column

You may want to decrypt an encrypted application table column, for example after encrypting the tablespace that contains the application table.

- To decrypt a table column, use the `ALTER TABLE MODIFY` command with the `DECRYPT` clause.

  The following example decrypts the `first_name` column in the `employee` table.

  ```
  ALTER TABLE employee MODIFY (first_name DECRYPT);
  ```

5.6 Creating an Index on an Encrypted Column

You can create an index on an encrypted column.

The column being indexed must be encrypted without salt. If the column is encrypted with salt, then the `ORA-28338: cannot encrypt indexed column(s) with salt` error is raised.

- To create an index on an encrypted column, use the `CREATE INDEX` statement with the `ENCRYPT NO SALT` clause.

  The following example shows how to create an index on a column that has been encrypted without salt.

```
CREATE TABLE employee (
   first_name VARCHAR2(128),
   last_name VARCHAR2(128),
   empID NUMBER ENCRYPT NO SALT,
   salary NUMBER(6) ENCRYPT);

CREATE INDEX employee_idx on employee (empID);
```

5.7 Adding Salt to an Encrypted Column

Salt, which is a random string added to data before encryption, is a way to strengthen the security of encrypted data.

Salt ensures that the same plaintext data does not always translate to the same encrypted text. Salt removes the one common method that intruders use to steal data, namely, matching patterns of encrypted text. Adding salt requires an additional 16 bytes of storage per encrypted data value.

- To add or remove salt from encrypted columns, use the `ALTER TABLE MODIFY` SQL statement.

  For example, suppose you want to encrypt the `first_name` column using salt. If the `first_name` column was encrypted without salt earlier, then the `ALTER TABLE MODIFY` statement reencrypts it using salt.

  ```
  ALTER TABLE employee MODIFY (first_name ENCRYPT SALT);
  ```

5.8 Removing Salt from an Encrypted Column

You can use the `ALTER TABLE` SQL statement to remove salt from an encrypted column.

- To remove salt from an encrypted column, use the `ENCRYPT NO SALT` clause in the `ALTER TABLE` SQL statement.

  For example, suppose you wanted to remove salt from the `first_name` column. If you must index a column that was encrypted using salt, then you can use this statement to remove the salt before indexing.

  ```
  ALTER TABLE employee MODIFY (first_name ENCRYPT NO SALT);
  ```

5.9 Changing the Encryption Key or Algorithm for Tables with Encrypted Columns

You can use the `ALTER TABLE` SQL statement to change the encryption key or algorithm used in encrypted columns.

Each table can have only one TDE table key for its columns. You can regenerate the TDE table key with the `ALTER TABLE` statement. This process generates a new key, decrypts the data in the table using the previous key, reencrypts the data using the new key, and then updates the table metadata with the new key information. You can also use a different encryption algorithm for the new TDE table key.

- To change the encryption key or algorithm for tables that contain encrypted columns, use the `ALTER TABLE` SQL statement with the `REKEY` or `REKEY USING` clause.

  For example:

  ```
  ALTER TABLE employee REKEY;
  ```

The following example regenerates the TDE table key for the `employee` table by using the `AES256` algorithm.

```
ALTER TABLE employee REKEY USING 'AES256';
```

5.10 Migrating the Algorithm to the Latest Supported Algorithm for Tables

Re-encrypting an already encrypted table column enables you to migrate an earlier algorithm (for example, `3DES168`) to the latest supported algorithm.

1. Query the `DBA_ENCRYPTED_COLUMNS` data dictionary viiew to find the encryption algorithms that the tables currently use.

   ```
   SELECT TABLE_NAME, ENCRYPTION_ALG
   FROM DBA_ENCRYPTED_COLUMNS
   WHERE ENCRYPTION_ALG = '3 Key Triple DES 168 bits key';
   ```

   Output similar to the following appears:

   ```
   TABLE_NAME           ENCRYPTION_ALG
   -------------------- ------------------------------
   EMPS                 3 Key Triple DES 168 bits key
   ```

2. Re-encrypt each table that is listed in the output.

   For example, to migrate to the `AES256` algorithm:

   ```
   ALTER TABLE TEST REKEY USING 'AES256';
   ```

   You can write a PL/SQL procedure to put all the tables in a list, and then iterate over the list and re-encrypt the tables in the procedure, or you can manually re-encrypt all tables.

3. To verify the re-encryption:

   The following example queries the `DBA_ENCRYPTED_COLUMNS` data dictionary view:

   ```
   SELECT TABLE_NAME, ENCRYPTION_ALG
   FROM DBA_ENCRYPTED_COLUMNS
   WHERE TABLE_NAME = 'EMPS';

   TABLE_NAME           ENCRYPTION_ALG
   -------------------- ------------------------------
   EMPS                 AES 256 bits key
   ```

6 Encryption Conversions for Tablespaces and Databases

You can perform encryption operations on both offline and online tablespaces and databases.

6.1 About Encryption Conversion for Tablespaces and Databases

The `CREATE TABLESPACE` SQL statement can be used to create a new, encrypted tablespace. `ALTER TABLESPACE` can encrypt existing tablespaces.

In addition to encrypting new and existing tablespaces, you can encrypt full databases, which entails the encryption of the Oracle-managed tablespaces (in this release, the `SYSTEM`, `SYSAUX`, `TEMP`, and `UNDO` tablespaces). An Oracle-supplied tablespace contains information necessary for the correct functioning (confidentiality, integrity, and availability) of the database system. This information includes the system's data dictionary, the system's temporary sort area, the system's undo segment, and the system's auxiliary data. This information is only expected to be updated internally by the Oracle database server itself, and does not normally be updated directly by users.

To encrypt a full database, you use the `ALTER TABLESPACE` statement, not `ALTER DATABASE`, to encrypt the Oracle-managed tablespaces.

The following table compares the differences between an offline and an online encryption conversion of tablespaces and databases.

Table 6-1 Offline and Online Tablespace and Database Encryption Conversions

Functionality	Offline Conversion	Online Conversion
Release with minimum conversion capability	Oracle Database 11g release 2 (11.2)	Oracle Database 12c release 2 (12.2) and later
What can be backported?	The ability to encrypt or decrypt a data file with the `AES128` algorithm (using `ALTER DATABASE DATAFILE` *`data_file`* `ENCRYPT` or `DECRYPT`) can be used in Oracle Database releases 12.1.0.2 and 11.2.0.4.	No
Algorithms supported	All symmetric encryption algorithms that TDE supports.	All symmetric encryption algorithm that TDE supports.
When can the conversion be run?	When the tablespace is offline or the database is in the mount stage.	When the tablespace is online and database is open in read/write mode.
Is auxiliary space required for the conversion?	No	Yes.

Table 6-1 (Cont.) Offline and Online Tablespace and Database Encryption Conversions

Functionality	Offline Conversion	Online Conversion
Oracle Data Guard conversion guidelines	Convert both the primary and standby manually. Convert the standby first and then switch over to minimum downtime	After you convert the primary, the standby conversion takes place automatically. You cannot perform an online conversion directly on the standby.
Encrypt the `SYSTEM`, `SYSAUX`, and `UNDO` tablespaces (database conversion)	Oracle Database 12c release 2 (12.2) and later only. You must set `COMPATIBILITY` to `12.2.0.0`. Only auto-login keystores are supported.	Oracle Database 12c release 2 (12.2) and later only. You must set `COMPATIBILITY` to `12.2.0.0`.
Can an existing `TEMP` tablespace be converted?	No, but you can create an encrypted `TEMP` tablespace in Oracle Database 12c release 2 (12.2) and later, make it the default temporary tablespace, and then drop the original `TEMP` tablespace.	No, but you can create an encrypted `TEMP` tablespace in Oracle Database 12c release 2 (12.2) and later, make it the default temporary tablespace, and then drop the original `TEMP` tablespace.
Can an existing tablespace be decrypted?	You only can decrypt a tablespace or data file that was previously encrypted by an offline encrypt operation. Oracle does not recommend that you decrypt the `UNDO` tablespace once it is encrypted.	Yes, but Oracle does not recommend that you decrypt the `UNDO` tablespace once it is encrypted.
Can encryption keys be rekeyed?	No, but after the tablespace is encrypted, you can then use online conversion to rekey in Oracle Database 12c release 2 (12.2) compatibility.	Yes
Can encryption operations be run in parallel?	You can run parallel encryption conversions at the data file level with multiple user sessions running.	You can run parallel encryption conversions at the tablespace level with multiple user sessions running.
What to do if an encryption conversion SQL statement fails to complete?	Re-issue the encryption or decryption SQL statement to ensure that all the data files within the tablespace are consistently either encrypted or decrypted.	Rerun the SQL statement but use the `FINISH` clause.

Related Topics

- Supported Encryption and Integrity Algorithms
  Oracle supports the AES, ARIA and DES algorithms.
- About Offline Encryption of Existing Tablespaces
  You can encrypt or decrypt an existing data file of a user tablespace when the tablespace is offline or when the database is not open.
- About Encryption Conversions for Existing Online Tablespaces
  You can encrypt, decrypt, or rekey existing user tablespaces, and the `SYSTEM`, `SYSAUX`, and `UNDO` tablespace when they are online.

6.2 Impact of a Closed TDE Keystore on Encrypted Tablespaces

A TDE keystore can be closed or migrated when an Oracle-managed tablespace is encrypted, and the database system itself must be shut down to disallow operations on an Oracle-managed tablespace.

A closed TDE keystore has no impact on operations that involve an encrypted Oracle-managed tablespace (in this release, the `SYSTEM`, `SYSAUX`, `TEMP`, and `UNDO` tablespaces). This enables operations that are performed by background processes (for example, the log writer) to continue to work on these tablespaces while the TDE keystore is closed. If you want to disallow operations on an encrypted Oracle-managed tablespace, then you must shut down the database.

With regard to user-created tablespaces, a closed TDE keystore causes operations such as rotating a key or decrypting the tablespace to fail with an `ORA-28365 wallet is not open` error, just as it did in earlier releases. If you want to disallow operations on the user-created tablespace, then close the TDE keystore (or shut down the database).

User-created data can be copied into an encrypted Oracle-managed tablespace (for example, by an internal process such as `DBMS_STATS` statistics gathering) from a user-created tablespace while the TDE keystore is open. Closing the keystore does not prevent a user from viewing this data afterward, when the TDE keystore is in the `CLOSED` state at the time that you query the `V$ENCRYPTION_WALLET` view. Access to the original data by attempting to query an encrypted user-created tablespace will fail, resulting in an `ORA-28365 wallet is not open` error.

Table 6-2 describes the operations that are necessary to disallow or allow operations on encrypted data in user-created tablespaces and Oracle-managed tablespaces. For example, in the first scenario, both the user-created tablespaces and the Oracle-managed tablespaces are encrypted. In this case, for the encrypted data in the encrypted user-created tablespace, an administrator can close or open keystores, and shut down and open a database with an encrypted user-created tablespace. When an encrypted Oracle-managed tablespace is configured, the administrator can disallow operations by shutting down the database, and can allow operations by starting up in mount mode, opening the TDE keystore, and then opening the database. (It is necessary to open the TDE keystore *before* opening the database because the system may need the TDE master encryption key to decrypt the bootstrap dictionary tables, which are located in the encrypted Oracle-managed tablespace.) The N/A flags in this table refer to non-encrypted data, which you can always operate on, unless the instance is shut down.

Table 6-2 Necessary Commands to Disallow or Allow Operations on Encrypted Data

Tablespace Encryption Scenarios	Commands to Disallow Operations on Encrypted User-Created Tablespace Data	Commands to Disallow Operations on Encrypted Oracle-Managed Tablespace Data	Commands to Allow Operations on Encrypted User-Created Tablespace Data	Commands to Allow Operations on Encrypted Oracle-Managed Tablespace Data
Both user-created and Oracle-managed tablespaces encrypted	• `ADMINISTER KEY MANAGEMENT SET KEYSTORE CLOSE IDENTIFIED BY` *`password`*`;` • `SHUTDOWN`	`SHUTDOWN`	• `STARTUP MOUNT;` • `ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN IDENTIFIED BY` *`password`*`;` • `ALTER DATABASE OPEN`	• `STARTUP MOUNT;` • `ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN IDENTIFIED BY` *`password`*`;` • `ALTER DATABASE OPEN`

Table 6-2 (Cont.) Necessary Commands to Disallow or Allow Operations on Encrypted Data

Tablespace Encryption Scenarios	Commands to Disallow Operations on Encrypted User-Created Tablespace Data	Commands to Disallow Operations on Encrypted Oracle-Managed Tablespace Data	Commands to Allow Operations on Encrypted User-Created Tablespace Data	Commands to Allow Operations on Encrypted Oracle-Managed Tablespace Data
User tablespace encrypted; Oracle-managed tablespace not encrypted	• `ADMINISTER KEY MANAGEMENT SET KEYSTORE CLOSE IDENTIFIED BY` *`password`*`;` • `SHUTDOWN`	N/A	• `STARTUP MOUNT;` • `ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN IDENTIFIED BY` *`password`*`;` • `ALTER DATABASE OPEN`	N/A
User tablespace not encrypted; Oracle-managed tablespace encrypted	N/A	`SHUTDOWN`	N/A	• `STARTUP MOUNT;` • `ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN IDENTIFIED BY` *`password`*`;` • `ALTER DATABASE OPEN`
Neither user nor Oracle-managed tablespaces encrypted	N/A	N/A	N/A	N/A

6.3 Restrictions on Using Transparent Data Encryption Tablespace Encryption

You should be aware of restrictions on using Transparent Data Encryption when you encrypt a tablespace.

Note the following restrictions:

- Transparent Data Encryption (TDE) tablespace encryption encrypts or decrypts data during read and write operations, as opposed to TDE column encryption, which encrypts and decrypts data at the SQL layer. This means that most restrictions that apply to TDE column encryption, such as data type restrictions and index type restrictions, do not apply to TDE tablespace encryption.
- To perform import and export operations, use Oracle Data Pump.

Related Topics

- Encrypting an Unencrypted Column
  You can use the `ALTER TABLE MODIFY` statement to encrypt an existing unencrypted column.
- Data Types That Can Be Encrypted with TDE Column Encryption
  Oracle Database supports a specific set of data types that can be used with TDE column encryption.
- *Oracle Database Utilities*

6.4 Creating an Encrypted New Tablespace

When you create a new tablespace, you can configure its encryption settings during the creation process.

6.4.1 Step 1: Set the COMPATIBLE Initialization Parameter for Tablespace Encryption

You must set the `COMPATIBLE` initialization parameter before creating an encrypted tablespace.

6.4.1.1 About Setting the COMPATIBLE Initialization Parameter for Tablespace Encryption

A minimum `COMPATIBLE` initialization parameter setting of `11.2.0.0` enables the full set of tablespace encryption features.

Setting the compatibility to `11.2.0.0` enables the following functionality:

- The `11.2.0.0` setting enables the database to use any of the four supported algorithms for data encryption (`3DES168`, `AES128`, `AES192`, and `AES256`).
- The `11.2.0.0` setting enables the migration of a key from a TDE wallet to an external keystore (ensure that the TDE master encryption key was configured for the external keystore)
- The `11.2.0.0` setting enables rekeying the TDE master encryption key

Be aware that once you set the `COMPATIBLE` parameter to `11.2.0.0`, the change is irreversible. To use tablespace encryption, ensure that the compatibility setting is at the minimum, which is `11.2.0.0`.

Related Topics

- *Oracle Database SQL Language Reference*
- *Oracle Database Administrator's Guide*

6.4.1.2 Setting the COMPATIBLE Initialization Parameter for Tablespace Encryption

To set the `COMPATIBLE` initialization parameter, you must edit the initialization parameter file for the database instance.

1. Connect to the united mode CDB root or isolated mode PDB as a user who has been granted administrative privileges.
2. Check the current setting of the `COMPATIBLE` parameter.

   For example:

   ```
   SHOW PARAMETER COMPATIBLE

   NAME                                 TYPE        VALUE
   ------------------------------------ ----------- ------------------------------
   compatible                           string      12.2.0.0
   noncdbcompatible                     BOOLEAN     FALSE
   ```

3. If you must change the `COMPATIBLE` parameter, then complete the remaining steps in this procedure.

The value should be 11.2.0.0 or later.

4. From the command line, locate the initialization parameter file for the database instance.
   - **UNIX systems:** This file is in the *ORACLE_HOME*/dbs directory and is named init*ORACLE_SID*.ora (for example, initmydb.ora).
   - **Windows systems:** This file is in the *ORACLE_HOME*\database directory and is named init*ORACLE_SID*.ora (for example, initmydb.ora).
5. Edit the initialization parameter file to use the new COMPATIBLE setting.

   For example:

   ```
   COMPATIBLE=20.1.0.0
   ```

6. Connect as a user who has the SYSDBA administrative privilege, and then restart the database.
   - From the CDB root:

     ```
     SHUTDOWN IMMEDIATE
     STARTUP
     ```

   - From a PDB:

     ```
     ALTER PLUGGABLE DATABASE pdb_name CLOSE IMMEDIATE;
     ALTER PLUGGABLE DATABASE pdb_name OPEN;
     ```

7. If tablespace encryption is in use, then open the keystore at the database mount. The keystore must be open before you can access data in an encrypted tablespace.

   For example:

   ```
   STARTUP MOUNT;
   ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN IDENTIFIED BY keystore_password|EXTERNAL
   STORE;
   ALTER DATABASE OPEN;
   ```

6.4.2 Step 2: Set the TDE Master Encryption Key

You must ensure that you have configured TDE by setting the WALLET_ROOT and TDE_CONFIGURATION parameters.

- Set the TDE master encryption key either in the Oracle wallet, or in Oracle Key Vault. The commands for united and isolated PDBs are the same.

6.4.3 Step 3: Create the Encrypted Tablespace

After you have set the COMPATIBLE initialization parameter, you are ready to create the encrypted tablespace.

6.4.3.1 About Creating Encrypted Tablespaces

To create an encrypted tablespace, you can use the CREATE TABLESPACE SQL statement.

You must have the CREATE TABLESPACE system privilege to create an encrypted tablespace.

You can query the ENCRYPTED column of the DBA_TABLESPACES and USER_TABLESPACES data dictionary views to verify if a tablespace was encrypted.

6.4.3.2 Creating an Encrypted Tablespace

To create an encrypted tablespace, you must use the `CREATE TABLESPACE` statement with the `ENCRYPTION USING` clause.

Run the `CREATE TABLESPACE` statement, using its encryption clauses.
For example:

```
CREATE TABLESPACE encrypt_ts
  DATAFILE '$ORACLE_HOME/dbs/encrypt_df.dbf' SIZE 1M
  'ENCRYPTION ENCRYPT';
```

In this specification:

- `'ENCRYPTION ENCRYPT'` uses the default for encryption algorithm (`AES256`) and cipher mode (XTS), unless specified otherwise. The `ENCRYPT` clause encrypts the tablespace. Enclose this setting in single quotation marks (' '). The key lengths are included in the names of the algorithms.
- If `TABLESPACE_ENCRYPTION` is set to `AUTO_ENABLE`, the `'ENCRYPTION` *`[using algorithm]`* `ENCRYPT'` keywords can be omitted.

Related Topics

- Supported Encryption and Integrity Algorithms
  Oracle supports the AES, ARIA and DES algorithms.

6.4.3.3 Example: Creating an Encrypted Tablespace That Uses AES192

You can use the `CREATE TABLESPACE` SQL statement to create an encrypted tablespace.

Example 6-1 creates a tablespace called `securespace_1` that is encrypted using the `AES192` algorithm.

Example 6-1 Creating an Encrypted Tablespace That Uses AES192

```
CREATE TABLESPACE securespace_1
DATAFILE '/home/user/oradata/secure01.dbf'
SIZE 150M
ENCRYPTION USING 'AES192' ENCRYPT;
```

6.4.3.4 Example: Creating an Encrypted Tablespace That Uses the Default Algorithm

You can use the `CREATE TABLESPACE` SQL statement to create an encrypted tablespace that uses the default algorithm.

Example 6-2 creates a tablespace called `securespace_2`. Because no encryption algorithm is specified, the default encryption algorithm (`AES256`) is used.

Example 6-2 Creating an Encrypted Tablespace That Uses the Default Algorithm

```
CREATE TABLESPACE securespace_2
DATAFILE '/home/user/oradata/secure01.dbf'
SIZE 150M
ENCRYPTION ENCRYPT;
```

6.5 Setting the Tablespace Encryption Default Algorithm

The `TABLESPACE_ENCRYPTION_DEFAULT_ALGORITHM` applies to specific encryption scenarios.

These scenarios are as follows:

- In Oracle Database 23ai, `AES256` (with XTS cipher mode) is the default because it has the highest resistance against post-quantum attacks. It is not recommended to use shorter data encryption keys.
- Encryption commands that do not allow to specify the encryption algorithm
- New tablespaces that are created without the encryption syntax
- The encryption algorithm for the `SYSTEM` tablespace

`TABLESPACE_ENCRYPTION_DEFAULT_ALGORITHM` only becomes effective if it is set before the first `SET KEY` operation with Oracle Key Vault, or the `CREATE KEYSTORE` command for wallet-based TDE configuration.

`TABLESPACE_ENCRYPTION_DEFAULT_ALGORITHM` applies to both offline and online tablespace encryption operations. It also applies to future encrypted tablespaces, if `TABLESPACE_ENCRYPTION` has been set appropriately. In a multitenant environment, you can set `TABLESPACE_ENCRYPTION_DEFAULT_ALGORITHM` in the CDB root or in individual PDBs.

- Enter the following `ALTER SYSTEM` statement:

  ```
  ALTER SYSTEM SET TABLESPACE_ENCRYPTION_DEFAULT_ALGORITHM = value SCOPE=BOTH;
  ```

  In this specification, *`value`* can be one of the following encryption algorithms: `AES128`, `AES192`, `AES256`, `3DES168`, `ARIA128`, `ARIA192`, or `ARIA256`. The default encryption algorithm is `AES256`. In FIPS mode, this parameter can take only `AES128`, `AES192`, and `AES256`.

> **Note:**
>
> Starting with Oracle Database 23ai, the decryption libraries for the GOST and SEED algorithms are deprecated and the GOST and SEED algorithms are desupported and removed. The GOST and SEED decryption libraries are deprecated. Both are removed on HP Itanium platforms.
> GOST 28147-89 has been deprecated by the Russian government, and SEED has been deprecated by the South Korean government. If you need South Korean government-approved TDE cryptography, then use ARIA instead. If you are using GOST 28147-89, then you must decrypt and encrypt with another supported TDE algorithm. The decryption algorithms for GOST 28147-89 and SEED are included with Oracle Database 23ai, but are deprecated, and the GOST encryption algorithm is desupported with Oracle Database 23ai. If you are using GOST or SEED for TDE encryption, then Oracle recommends that you perform an online rekey operation before upgrading to Oracle Database 23ai. However, with the exception of the HP Itanium platform, the GOST and SEED decryption libraries are available with Oracle Database 23ai, so you can also decrypt after upgrading.

6.6 Encrypting Future Tablespaces

You can configure Oracle Database to automatically encrypt future tablespaces that you will create.

6.6.1 About Encrypting Future Tablespaces

The `ENCRYPT_NEW_TABLESPACES` dynamic database initialization parameter controls if future tablespaces are encrypted.

By default, all Oracle Cloud databases are encrypted. If you install an off-the-shelf application into such a database, its installation scripts most likely do not have the encryption syntax. In this case, because `ENCRYPT_NEW_TABLESPACES` is set to `CLOUD_ONLY`, those tablespaces would be created encrypted regardless.

> **Note:**
>
> Starting with Oracle Database 23ai, the `ENCRYPT_NEW_TABLESPACES` initialization parameter is deprecated. Oracle recommends that you use the initialization parameter `TABLESPACE_ENCRYPTION`.

In an Oracle Cloud environment, the following scenarios may occur when you create encrypted tablespaces in Oracle Cloud and on-premises environments:

- You create a database in Oracle Cloud; the tablespaces are encrypted because the `TABLESPACE_ENCRYPTION` parameter is set to `AUTO_ENABLE`. However, you may not have the intention or even an Advanced Security Option license to bring the encrypted database back on premises. For this use case, Oracle Recovery Manager (Oracle RMAN) provides the option to duplicate or restore `AS DECRYPTED`.
- You create a hybrid environment where the primary database is on premises and the standby database is on Oracle Cloud. If a switchover operation takes place, then the new primary is on Oracle Cloud. For example, suppose you do not have an Advanced Security Option (ASO) license, and you have an automatically encrypted tablespace in the Oracle Cloud. The standby database on premises is also automatically encrypted, then the standby database on premises can set `TABLESPACE_ENCRYPTION` to `DECRYPT_ONLY`. See the Hybrid Oracle Data Guard without Transparent Data Encryption (TDE) License Oracle Database Product Management Youtube video.

Related Topics

- Encryption of Tablespaces in an Oracle Data Guard Environment
  You can control tablespace encryption in the primary and standby databases in an Oracle Data Guard environment.

6.6.2 Setting Future Tablespaces to be Encrypted

You can set the `TABLESPACE_ENCRYPTION` database initialization parameter to automatically encrypt new tablespaces.

> **Note:**
>
> Starting with Oracle Database 19.16, the `ENCRYPT_NEW_TABLESPACES` has been deprecated.
> Oracle recommends that you use the initialization parameter `TABLESPACE_ENCRYPTION`, which is new for Oracle Database 23ai.

- Enter the following `ALTER SYSTEM` statement:

  ```
  ALTER SYSTEM SET TABLESPACE_ENCRYPTION = value SCOPE = SPFILE SID = '*';
  ```

  In this specification, *`value`* can be:

  - `AUTO_ENABLE` automatically encrypts the tablespace using the tablespace encryption default algorithm and cipher mode (XTS if `compatible` is set to 23 or higher) if you do not specify the `ENCRYPTION` clause of the `CREATE TABLESPACE` SQL statement, or to the algorithm specified by the `TABLESPACE_ENCRYPTION_DEFAULT_ALGORITHM` dynamic parameter.
  - `DDL` encrypts the tablespace using the specified setting of the `ENCRYPTION` clause of `CREATE TABLESPACE`, for both Oracle Cloud and on-premises environments.

Related Topics

- Encryption of Tablespaces in an Oracle Data Guard Environment
  You can control tablespace encryption in the primary and standby databases in an Oracle Data Guard environment.
- *Oracle Database Reference*

6.7 Encrypted Sensitive Credential Data in the Data Dictionary

You can encrypt sensitive credential data in the `SYS.LINK$` and `SYS.SCHEDULER$_CREDENTIAL` system tables.

By default, the credential data in the `SYS.LINK$` and `SYS.SCHEDULER$_CREDENTIAL` system tables is obfuscated. However, because of the availability of many types of de-obfuscation algorithms, Oracle recommends that you encrypt this sensitive credential data. To check the status the data dictionary credentials, you can query the `DICTIONARY_CREDENTIALS_ENCRYPT` data dictionary view.

The encryption of sensitive credential data in these two system tables uses Transparent Data Encryption. To encrypt credential data, you do not need an Oracle Advanced Security Option license, but you must be granted the `SYSKM` administrative privilege and the database must have a fully functional Transparent Data Encryption setup. Encryption of credential data uses the AES256 algorithm.

Related Topics

- *Oracle Database Security Guide*

6.8 Offline Encryption of Existing Tablespaces

You can perform offline encryption by using the `ALTER TABLESPACE` or `ALTER DATABASE DATAFILE SQL` statements with the `ENCRYPT` or `DECRYPT` clauses.

6.8.1 About Offline Encryption of Existing Tablespaces

You can encrypt or decrypt an existing data file of a user tablespace when the tablespace is offline or when the database is not open.

There are two options for performing the encryption or decryption:

- **Offline encryption of the data file level:** This type does not have the option to specify an encryption algorithm. It uses the default encryption algorithm. For example:

  ```
  ALTER DATABASE DATAFILE 'path_to_data_file.dbf' ENCRYPT;
  ```

- **Offline encrypting a tablespace:** Use the following syntax. If you omit the `[USING 'algorithm']`, then the default encryption algorithm is used.

  ```
  ALTER TABLESPACE tablespace_name ENCRYPTION OFFLINE [USING 'algorithm']
  ENCRYPT;
  ```

Use the offline encryption method if you do not plan to change the compatibility of your databases from Oracle Database 11c release 2 (11.2) or Oracle Database 12c release 1 (12.1) to release 19c, which is irreversible. The offline encryption method is also useful if you want to quickly make use of Transparent Data Encryption before you upgrade this database to release 19c. You can both encrypt and decrypt offline tablespaces.

Note the following:

- If you want to encrypt the Oracle Database-supplied tablespaces (`SYSTEM`, `SYSAUX`, and `UNDO`) using the offline conversion method, then you must use the method that is recommended when you encrypt an existing database with offline conversion.
- You can use the online method to rekey, change the encryption algorithm, or decrypt a tablespace that was previously encrypted with the offline method.
- If you have configured Oracle Data Guard, then you can minimize downtime by encrypting the tablespaces on the standby first, switching over to the primary, and then performing an offline encryption the tablespaces on the new standby database. Offline encryption (both on the data file and tablespace level) are is performed on the standby first. Online encryption is an Oracle Data Guard transaction, and as such, it is replayed on the standby database.
- You can use the `USING ... ENCRYPT` clause to specify an encryption algorithm. Supported algorithms are AES and ARIA with 128, 192, and 256 bits key length. To check the encryption key, query the `ENCRYPTIONALG` column in the `V$DATABASE_KEY_INFO` view.

> **Note:**
>
> Starting with Oracle Database 23ai, the Transparent Data Encryption (TDE) decryption libraries for the GOST and SEED algorithms are deprecated, and encryption to GOST and SEED are desupported. Starting with Oracle Database 23ai, the Transparent Data Encryption (TDE) encryption libraries for the GOST and SEED algorithms are desupported and removed. The GOST and SEED decryption libraries are deprecated. Both are removed on HP Itanium platforms. GOST 28147-89 has been deprecated by the Russian government, and SEED has been deprecated by the South Korean government. If you need South Korean government-approved TDE cryptography, then use ARIA instead. If you are using GOST 28147-89, then you must decrypt and encrypt with another supported TDE algorithm. The decryption algorithms for GOST 28147-89 and SEED are included with Oracle Database 23ai, but are deprecated, and the GOST encryption algorithm is desupported with Oracle Database 23ai. If you are using GOST or SEED for TDE encryption, then Oracle recommends that you perform an online rekey operation before upgrading to Oracle Database 23ai. However, with the exception of the HP Itanium platform, the GOST and SEED decryption libraries are available with Oracle Database 23ai, so you can also decrypt after upgrading.

- To set the default encryption algorithm for future offline tablespace encryption operations, set the `TABLESPACE_ENCRYPTION_DEFAULT_ALGORITHM` dynamic parameter.
- You can use the `ALTER TABLESPACE` statement to convert offline tablespaces in parallel by using multiple foreground sessions to encrypt different data files.
- If you are using Oracle Data Guard, you can minimize the downtime by encrypting the tablespaces on the standby first, switching over, and then encrypting the tablespaces on the original primary next.
- For Oracle Database 11g release 2 (11.2.0.4) and Oracle Database 12c release 1 (12.1.0.2), you cannot perform an offline encryption of the `SYSTEM` and `SYSAUX` tablespaces. For releases earlier than Oracle Database 19c, you cannot encrypt the `SYSTEM`, `SYSAUX`, `TEMP`, and `UNDO` tablespaces. Also, Oracle does not recommend encrypting offline the `UNDO` tablespace in these releases. Doing so prevents the keystore from being closed, and this prevents the database from functioning. In addition, encrypting the `UNDO` tablespace while the database is offline is not necessary because all undo records that are associated with any encrypted tablespaces are already automatically encrypted in the `UNDO` tablespace. If you want to encrypt the `TEMP` tablespace, you must drop and then recreate it as encrypted.

Related Topics

- Encrypting an Existing Database with Offline Conversion
  When you encrypt an existing database with offline conversion, for the Oracle-managed tablespaces, you do not specify an encryption algorithm.
- Supported Encryption and Integrity Algorithms
  Oracle supports the AES, ARIA and DES algorithms.
- Setting the Tablespace Encryption Default Algorithm
  The `TABLESPACE_ENCRYPTION_DEFAULT_ALGORITHM` applies to specific encryption scenarios.

6.8.2 Encrypting an Existing User-Defined Tablespace with Offline Conversion

To encrypt an existing tablespace with offline conversion, you can use the `ALTER TABLESPACE` SQL statement with the `OFFLINE` and `ENCRYPT` clauses.

The procedure that is described in this section applies to the case where you want to encrypt individual user-created tablespaces within a database. These tablespaces can be encrypted offline. However, the Oracle Database-supplied `SYSTEM` and `UNDO` tablespaces cannot be brought offline. If you want to encrypt the tablespaces offline, then you must use the method that is described in Encrypting an Existing Database with Offline Conversion.

1. Connect to the united mode CDB root or isolated mode PDB as a user who has been granted administrative privileges.

   You must have the `SYSDBA` administrative privilege to work with the `SYSTEM` and `SYSAUX` tablespaces. Otherwise, connect with the `SYSKM` administrative privilege.

2. Bring the tablespace offline.

   ```
   ALTER TABLESPACE users OFFLINE NORMAL;
   ```

3. Back up the tablespace.

   The offline conversion method does not use auxiliary disk space or files, and it operates directly in-place to the data files. Therefore, you should perform a full backup of the user tablespace before converting it offline.

4. As a user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege, open the TDE wallet.

   For example:

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

5. Encrypt the tablespace.

   For example, to encrypt an entire tablespace, include its data files:

   ```
   ALTER TABLESPACE users1 ENCRYPTION OFFLINE ENCRYPT;
   ```

   This example encrypts the tablespace with the default encryption algorithm, `AES256`. To use a different encryption algorithm, enter a statement similar to the following:

   ```
   ALTER TABLESPACE users2 ENCRYPTION OFFLINE USING 'AES128' ENCRYPT;
   ```

   To encrypt individual data files within a tablespace, use the `ALTER DATABASE DATAFILE` SQL statement. For example, to encrypt the data files `user_01.dbf` and `user_02.dbf`:

   ```
   ALTER DATABASE DATAFILE 'user_01.dbf' ENCRYPT;
   ALTER DATABASE DATAFILE 'user_02.dbf' ENCRYPT;
   ```

   In the same database session, these statements encrypt each of the data files in sequence, one after another. If you run each statement in its own database session, then they will be run in parallel.

   If the encryption process is interrupted, then rerun the `ALTER TABLESPACE` statement. The kinds of errors that you can expect in an interruption are general errors, such as file system or storage file system errors. The data files within the tablespace should be consistently encrypted. For example, suppose you offline a tablespace that has 10 files but for some reason, the encryption only completes for nine of the files, leaving one decrypted. Although

it is possible to bring the tablespace back online with such inconsistent encryption if the COMPATIBLE parameter is set to 12.2.0.0 or later, then it is not recommended to leave the tablespace in this state. If COMPATIBLE is less than 12.2.0.0, then it is not possible to bring the tablespace online if the encryption property is inconsistent across the data files.

6. Bring the tablespace back online or open the database.
   - To bring the tablespace back online:

     ```
     ALTER TABLESPACE users ONLINE;
     ```

   - To open a database in a non-multitenant environment:

     ```
     ALTER DATABASE OPEN
     ```

   - In a multitenant environment, you can encrypt a data file or tablespace with the offline method if the root is open and the PDB is not open. For example, for a PDB named hr_pdb:

     ```
     ALTER PLUGGABLE DATABASE hr_pdb OPEN
     ```

7. Perform a full backup of the converted tablespace.

Related Topics

- Supported Encryption and Integrity Algorithms
  Oracle supports the AES, ARIA and DES algorithms.

6.8.3 Decrypting an Existing Tablespace with Offline Conversion

To decrypt an existing tablespace with offline conversion, you can use the ALTER TABLESPACE SQL statement with the OFFLINE and DECRYPT clauses.

1. Connect to the united mode CDB root or isolated mode PDB as a user who has been granted the SYSDBA administrative privilege.

   You must have the SYSDBA administrative privilege to work with the SYSTEM and SYSAUX tablespaces. Otherwise, connect with the SYSKM administrative privilege.

2. Bring the tablespace offline.

   ```
   ALTER TABLESPACE users OFFLINE NORMAL;
   ```

3. As a user who has been granted the ADMINISTER KEY MANAGEMENT or SYSKM privilege, open the keystore.

   For example:

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

4. Run the ALTER TABLESPACE SQL statement to perform the decryption.

   For example, for a tablespace called users:

   ```
   ALTER TABLESPACE users ENCRYPTION OFFLINE DECRYPT;
   ```

   If the decryption process is interrupted, then rerun the ALTER TABLESPACE statement. The kinds of errors that you can expect in an interruption are general errors, such as file system or storage file system errors. The data files within the tablespace should be consistently decrypted. For example, suppose you offline a tablespace that has 10 files but for some reason, the decryption only completes for nine of the files, leaving one encrypted. Although it is possible to bring the tablespace back online with such inconsistent decryption if the COMPATIBLE parameter is set to 12.2.0.0 or later, then it is not recommended to leave the

tablespace in this state. If COMPATIBLE is less than 12.2.0.0, then it is not possible to bring the tablespace online if the encryption property is inconsistent across the data files.

5. Bring the tablespace online.

```
ALTER TABLESPACE users ONLINE;
```

6.9 Encryption Conversions for Existing Online Tablespaces

You can encrypt and decrypt an online existing tablespace by using the ALTER TABLESPACE SQL statement with the ONLINE and ENCRYPT or DECRYPT clauses.

6.9.1 About Encryption Conversions for Existing Online Tablespaces

You can encrypt, decrypt, or rekey existing user tablespaces, and the SYSTEM, SYSAUX, and UNDO tablespace when they are online.

However, you cannot encrypt, decrypt, or rekey a temporary tablespace online.

An online tablespace can be created by using the ONLINE clause of the CREATE TABLESPACE SQL statement. When you encrypt or rekey a tablespace online, the tablespace will have its own independent encryption keys and algorithms.

Note the following:

- If an offline tablespace has been encrypted, then you can rekey it online to use a different algorithm.
- You can encrypt multiple tablespaces online in parallel by using multiple foreground sessions to encrypt different tablespaces. Within each tablespace, the data files are encrypted sequentially.
- To set the default encryption algorithm for future online tablespace encryption operations, set the TABLESPACE_ENCRYPTION_DEFAULT_ALGORITHM dynamic parameter.
- If the conversion is interrupted, then you can resume the process by issuing the FINISH clause of the ALTER TABLESPACE SQL statement.
- A redo log is generated for each online tablespace conversion.
- Do not encrypt the SYSTEM and UNDO tablespaces concurrently with other tablespaces.
- You cannot use the transportable tablespace feature with Oracle Data Pump while you are encrypting a tablespace.
- You cannot run the ALTER TABLESPACE statement concurrently with the following features:
  - ADMINISTER KEY MANAGEMENT SET KEY SQL statement
  - FLASHBACK DATABASE SQL statement
- If you are using Oracle-managed files for the data files, then the encryption process rekeys the data files that are associated with the tablespace and then copies or moves them to the default Oracle-managed files location.
- You can add new files to the tablespace after you have encrypted it. Oracle Database reformats the new file with the new encryption key. Blocks will be encrypted using the new key.
- Previous operations that took place in the root or the PDB may require the control files to be cross-checked against the data dictionary before you can begin the online conversion process. An ORA-241 operation disallowed: control file is not yet checked

`against data dictionary` error may occur. To resolve this problem, restart the root or PDB, and then try issuing the online conversion commands again.

- For security reasons, once online conversion processes a data file, Oracle will zero out the original data file before deletion. This prevents the database from leaving ghost data on disk sectors. However, there is a known limitation that can occur if you are performing an online tablespace conversion at the same time that Oracle Recovery Manager (Oracle RMAN) is validating files. The online tablespace conversion processes each file one at a time. If Oracle RMAN is validating a file at the same time that it is being processed by the online tablespace conversion, then Oracle RMAN could report a corruption problem (`ORA-01578: ORACLE data block corrupted (file # , block # )`). It does this because it sees the blocks that comprise the file as zero. This is a false alarm and you can ignore the error. If this occurs, then try running the Oracle RMAN validation process again.

Related Topics

- Supported Encryption and Integrity Algorithms
  Oracle supports the AES, ARIA and DES algorithms.
- Setting the Tablespace Encryption Default Algorithm
  The `TABLESPACE_ENCRYPTION_DEFAULT_ALGORITHM` applies to specific encryption scenarios.

6.9.2 Encrypting an Existing Tablespace with Online Conversion

To encrypt an existing tablespace with online conversion, use `ALTER TABLESPACE` with the `ONLINE` and `ENCRYPT` clauses.

1. Connect to the united mode CDB root or isolated mode PDB as a user who has been granted the `SYSDBA` administrative privilege.

   You must have the `SYSDBA` administrative privilege to work with the `SYSTEM` and `SYSAUX` tablespaces. Otherwise, connect with the `SYSKM` administrative privilege.

2. Ensure that the `COMPATIBLE` initialization parameter is set to 12.2.0.0 or later.

   You can use the `SHOW PARAMETER` command to check the current setting of a parameter.

3. Ensure that the database is open in read-write mode.

   You can query the `STATUS` column of the `V$INSTANCE` dynamic view to find if a database is open and the `OPEN_MODE` column of the `V$DATABASE` view to find if it in read-write mode.

4. If necessary, open the database in read-write mode.

   ```
   ALTER DATABASE OPEN READ WRITE;
   ```

5. Ensure that the auxiliary space is at least the same size as the largest data file of this tablespace.

   This size requirement is because Oracle Database performs the conversion one file at a time. For example, if the largest data file of the tablespace is 32 GB, then ensure that you have 32 GB of auxiliary space. To find the space used by a data file, query the `BYTES` or `BLOCKS` column of the `V$DATAFILE` dynamic performance view.

6. As a user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege, create and open a master encryption key.

   For example:

   ```
   ADMINISTER KEY MANAGEMENT CREATE KEYSTORE 'TDE_wallet_location' IDENTIFIED BY
   TDE_wallet_password;
   ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN IDENTIFIED BY TDE_wallet_password;
   ADMINISTER KEY MANAGEMENT SET KEY IDENTIFIED BY TDE_wallet_password WITH BACKUP;
   ```

7. Run the `ALTER TABLESPACE` statement using the `ENCRYPTION` and `ENCRYPT` clauses to perform the encryption.

> **Note:**
>
> For the `SYSTEM` tablespace, you can use the `ENCRYPT` clause to encrypt the tablespace, but you cannot specify an encryption algorithm because it is encrypted with the existing database key the first time. After encrypting the `SYSTEM` tablespace, use the `REKEY` clause to specify the algorithm. See Rekeying an Existing Tablespace with Online Conversion for more information.

For example, for a non-Oracle managed files tablespace named `users`:

```
ALTER TABLESPACE users ENCRYPTION ONLINE USING 'AES256' ENCRYPT FILE_NAME_CONVERT =
('users.dbf', 'users_enc.dbf');
```

In this example:

- `ENCRYPTION ONLINE USING 'AES256' ENCRYPT` sets the statement to encrypt the tablespace `users` while it is online and assigns it the `AES256` encryption algorithm. If you omit the `USING` *`algorithm`* clause, then the default algorithm, `AES256`, is used.
- `FILE_NAME_CONVERT` specifies one or more pairs of data files that are associated with the tablespace. The first name in the pair is an existing data file, and the second name is for the encrypted version of this data file, which will be created after the `ALTER TABLESPACE` statement successfully runs. If the tablespace has more than one data file, then you must process them all in this statement. Note the following:
  - Separate each file name with a comma, including multiple pairs of files. For example:

    ```
    FILE_NAME_CONVERT = ('users1.dbf', 'users1_enc.dbf', 'users2.dbf',
    'users2_enc.dbf')
    ```

  - You can specify directory paths in the `FILE_NAME_CONVERT` clause. For example, the following clause converts and moves the matching files of the tablespace from the `dbs` directory to the `dbs/enc` directory:

    ```
    FILE_NAME_CONVERT = ('dbs', 'dbs/enc')
    ```

  - The `FILE_NAME_CONVERT` clause recognizes patterns. The following example converts the data files `users_1.dbf` and `users_2.dbf` to `users_enc1.dbf` and `users_enc2.dbf`:

    ```
    FILE_NAME_CONVERT = ('users', 'users_enc')
    ```

  - In an Oracle Data Guard environment, include the name of the standby database data file in the `FILE_NAME_CONVERT` settings.
  - If you are using Oracle-managed file mode, then the new file name is internally assigned, so this file name should not affect your site's file-naming standards. If you are using non-Oracle-managed file mode and if you omit the `FILE_NAME_CONVERT` clause, then Oracle Database internally assigns an auxiliary file name, and then later renames it back to the original name. This enables the encryption process to use the name that you had originally given the file to be encrypted. The renaming operation is effectively creating another copy of the file, hence it is slower than explicitly including the `FILE_NAME_CONVERT` clause. For better performance, include the `FILE_NAME_CONVERT` clause.

– You can find the data files for a tablespace by querying the V$DATAFILE or V$DATAFILE_HEADER dynamic views.

By default, data files are in the $ORACLE_HOME/dbs directory. If the data files are located there, then you do not have to specify a path.

After you complete the conversion, you can check the encryption status by querying the STATUS column of the V$ENCRYPTED_TABLESPACES dynamic view. The ENCRYPTIONALG column of this view shows the encryption algorithm that is used. If the conversion process was interrupted, then you can resume it by running ALTER TABLESPACE with the FINISH clause. For example, if the primary data file converts but the standby data file does not, then you can run ALTER TABLESPACE ... FINISH on the standby database for the standby data files.

Related Topics

- Best Practice after DBCA Creates an Encrypted Database
  After DBCA has created an encrypted stand-alone or Oracle Data Guard primary and standby database, you can implement Transparent Data Encryption (TDE) best practices.
- Setting the COMPATIBLE Initialization Parameter for Tablespace Encryption
  To set the COMPATIBLE initialization parameter, you must edit the initialization parameter file for the database instance.
- Finishing an Interrupted Online Encryption Conversion
  If an online encryption process is interrupted, then you can complete the conversion by rerunning the ALTER TABLESPACE statement using the FINISH clause.

6.9.3 Rekeying an Existing Tablespace with Online Conversion

To rekey an existing tablespace that is online, you can use the REKEY clause of the ALTER TABLESPACE SQL statement.

Before you perform a rekey operation, be aware of the following:

- You cannot rekey the TEMP tablespace. If you want to assign a different encryption algorithm to a TEMP tablespace, then drop TEMP and recreate it with the correct encryption algorithm.
- Do not perform an online tablespace rekey operation with a master key operation concurrently. To find if any tablespaces are currently being rekeyed, issue the following query to find the rekey status of encrypted tablespaces:

  ```
  SELECT TS#,ENCRYPTIONALG,STATUS FROM V$ENCRYPTED_TABLESPACES;
  ```

  A status of REKEYING means that the corresponding tablespace is still being rekeyed. Do not rekey the master key while this status is in effect.

To rekey an existing tablespace with online conversion:

1. Connect to the united mode CDB root or isolated mode PDB as a user who has been granted the SYSDBA administrative privilege.

   You must have the SYSDBA administrative privilege to work with the SYSTEM and SYSAUX tablespaces. Otherwise, connect with the SYSKM administrative privilege.

2. Ensure that the following requirements are met:
   - The COMPATIBLE initialization parameter is set to 12.2.0.0 or later.
   - The database is open and in read-write mode.
   - A master encryption key has been created and is open.

3. Query the KEY_VERSION and STATUS columns of the V$ENCRYPTED_TABLESPACES dynamic view to find the current status of the encryption algorithm used by the master encryption key.
4. Perform the rekey operation, based on the status returned by the V$ENCRYPTED_TABLESPACES dynamic view:
   - If the key version status of the tablespace is NORMAL, then specify the new algorithm of the online tablespace rekey.

     For example:

     ```
     ALTER TABLESPACE users ENCRYPTION USING 'AES192' REKEY FILE_NAME_CONVERT =
     ('users.dbf', 'users_enc.dbf');
     ```

   - If the key version status is ENCRYPTING, DECRYPTING, or REKEYING, then use the FINISH clause.

     For example:

     ```
     ALTER TABLESPACE users ENCRYPTION ONLINE FINISH REKEY FILE_NAME_CONVERT =
     ('users.dbf', 'users_enc.dbf');
     ```

5. If the ORA-00241 operation disallowed: control file inconsistent with data dictionary error appears, then restart the CDB root and then retry Step 4.

If the conversion process was interrupted, then you can resume it by running ALTER TABLESPACE with the FINISH clause.

Related Topics

- Encrypting an Existing Tablespace with Online Conversion
  To encrypt an existing tablespace with online conversion, use ALTER TABLESPACE with the ONLINE and ENCRYPT clauses.
- About Encryption Conversions for Existing Online Tablespaces
  You can encrypt, decrypt, or rekey existing user tablespaces, and the SYSTEM, SYSAUX, and UNDO tablespace when they are online.
- Finishing an Interrupted Online Encryption Conversion
  If an online encryption process is interrupted, then you can complete the conversion by rerunning the ALTER TABLESPACE statement using the FINISH clause.

6.9.4 Rekeying the SYSAUX and UNDO Tablespaces with Online Conversion

To rekey the SYSAUX and UNDO onlne tablespaces, you can use the REKEY clause of the ALTER TABLESPACE SQL statement.

1. Connect to the united mode CDB root or isolated mode PDB as a user who has been granted the SYSDBA administrative privilege.

   You must have the SYSDBA administrative privilege to work with the SYSTEM and SYSAUX tablespaces. Otherwise, connect with the SYSKM administrative privilege.
2. Ensure that the following requirements are met:
   - The COMPATIBLE initialization parameter is set to 12.2.0.0 or later.
   - The database is open and in read-write mode.
   - A master encryption key has been created and is open.

3. Depending on how the _TABLESPACE_ENCRYPTION_DEFAULT_ALGORITHM parameter is set, perform the rekey operation as follows:
   - If _TABLESPACE_ENCRYPTION_DEFAULT_ALGORITHM is set to AES256, then you only need the REKEY clause in the ALTER TABLESPACE statement for SYSAUX or UNDO. For example:

     ```
     ALTER TABLESPACE SYSAUX ENCRYPTION REKEY;
     ```

   - If _TABLESPACE_ENCRYPTION_DEFAULT_ALGORITHM is not set, then you must manually specify the AES256 algorithm. For example:

     ```
     ALTER TABLESPACE SYSAUX ENCRYPTION USING 'AES256' REKEY;
     ```

Related Topics

- About Encryption Conversions for Existing Online Tablespaces
  You can encrypt, decrypt, or rekey existing user tablespaces, and the SYSTEM, SYSAUX, and UNDO tablespace when they are online.

6.9.5 Decrypting an Existing Tablespace with Online Conversion

To decrypt an existing tablespace with online conversion, you can use the ALTER TABLESPACE SQL statement with DECRYPT clause.

1. Connect to the united mode CDB root or isolated mode PDB as a user who has been granted the SYSDBA administrative privilege.

   You must have the SYSDBA administrative privilege to work with the SYSTEM and SYSAUX tablespaces. Otherwise, connect with the SYSKM administrative privilege.

2. Ensure that the following requirements are met:
   - The COMPATIBLE initialization parameter is set to 12.2.0.0 or later.
   - The database is open and in read-write mode.
   - A master encryption key has been created and is open.
   - There is enough auxiliary space to complete the decryption.

3. Run the ALTER TABLESPACE SQL statement with the DECRYPT clause.

   For example:

   ```
   ALTER TABLESPACE users ENCRYPTION ONLINE DECRYPT FILE_NAME_CONVERT =
   ('users_enc.dbf', 'users.dbf');
   ```

   In this specification:

   - When you specify the files to decrypt, enter them in the reverse order in which they were originally encrypted. That is, first enter the name of the encrypted file (users_enc.dbf), followed by the data file (users.dbf).
   - Do not provide an algorithm key for the decryption.

If the conversion process was interrupted, then you can resume it by running ALTER TABLESPACE with the FINISH clause.

Related Topics

- Encrypting an Existing Tablespace with Online Conversion
  To encrypt an existing tablespace with online conversion, use `ALTER TABLESPACE` with the `ONLINE` and `ENCRYPT` clauses.
- Finishing an Interrupted Online Encryption Conversion
  If an online encryption process is interrupted, then you can complete the conversion by rerunning the `ALTER TABLESPACE` statement using the `FINISH` clause.

6.9.6 Finishing an Interrupted Online Encryption Conversion

If an online encryption process is interrupted, then you can complete the conversion by rerunning the `ALTER TABLESPACE` statement using the `FINISH` clause.

An interrupted encryption process (encryption, rekey, or decryption) can be, for example, an `ORA-28425: missing a valid FILE_NAME_CONVERT clause` error in the `FILE_NAME_CONVERT` clause of the `ALTER TABLESPACE` SQL statement. Other examples of interrupted processes are if the conversion skips a data file, which can happen if there is an error when an Oracle DataBase WRiter (DBWR) process offlines a data file, or if there is not enough space for the auxiliary file. The tablespace should be operational even if you do not rerun the `ALTER TABLESPACE` statement with the `FINISH` clause.

In addition to interrupted encryption processes, the tablespace encryption process can fail during the period when the status is `ENCRYPTING`. In this case, you can either decrypt the tablespace back to its original state, or you can resume the encryption by using the `ENCRYPTION ONLINE FINISH ENCRYPT` clause of `ALTER TABLEPSPACE`.

1. Connect to the united mode CDB root or isolated mode PDB as a user who has been granted the `SYSDBA` or `SYSKM` administrative privilege.

   You must have the `SYSDBA` administrative privilege to work with the `SYSTEM` and `SYSAUX` tablespaces. Otherwise, connect with the `SYSKM` administrative privilege.

2. Query the `V$ENCRYPTED_TABLESPACES` to check the `STATUS` column for the tablespace.

   If the `STATUS` column reports `ENCRYPTING`, `DECRYPTING`, or `REKEYING`, then re-run the `ALTER TABLESPACE` statement with the `FINISH` clause, as described in this procedure. If the `STATUS` reports `NORMAL`, then you can rerun `ALTER TABLESPACE` without the `FINISH` clause.

   You can find the tablespace name that matches the `TS#` and `TABLESPACE_NAME` columns by querying the `V$DATAFILE_HEADER` view.

3. If necessary query the following additional views to find information about the tablespace whose online conversion was interrupted:

   - `DBA_TABLESPACES` to find if the `STATUS` of the tablespace indicates if it is online or offline.
   - `V$ENCRYPTED_TABLESPACES` to find if the `STATUS` of the tablespace indicates if it is encrypted, and what the `KEY_VERSION` of the encryption key is.
   - `V$DATAFILE` and `V$DATAFILE_HEADER` to find the data files that are associated with a tablespace.

4. Run the `ALTER TABLESPACE` statement using the `FINISH` clause.

   Examples are as follows:

   - For an encryption operation:

```
ALTER TABLESPACE users ENCRYPTION ONLINE FINISH ENCRYPT FILE_NAME_CONVERT =
('users.dbf', 'users_enc.dbf');
```

- For a decryption operation:

  ```
  ALTER TABLESPACE users ENCRYPTION ONLINE FINISH DECRYPT FILE_NAME_CONVERT =
  ('users_enc.dbf', 'users.dbf');
  ```

  Note the order in which the files are specified: first, the name of the encrypted file, and then the name of the data file. (In the encryption operation, the name of the data file is specified first, followed by the name of the encrypted file.)

- For a rekey operation:

  ```
  ALTER TABLESPACE users ENCRYPTION ONLINE FINISH REKEY FILE_NAME_CONVERT =
  ('users.dbf', 'users_enc.dbf');
  ```

  You cannot specify an algorithm when you use the `FINISH` clause in an `ALTER TABLESPACE` statement.

5. To check the conversion, query the `STATUS` column of the `V$ENCRYPTED_TABLESPACES` view.

   The status should be `NORMAL`. In an Oracle Data Guard environment, if the database does not have `NORMAL` as the `STATUS`, then run the `ALTER TABLESPACE ... FINISH` statement on the primary or the standby data file that did not successfully convert.

6.10 Rekeying an Encrypted Tablespace

Rekying an already encrypted tablespace enables you to change the data encryption keys from one algorithm to another, for example from `AES128` to `AES256`.

1. Find the tablespaces that use the earlier algorithm.

   For example, to find tablespaces that use the `AES128`:

   ```
   SELECT TSVIEW.CON_ID, TSVIEW.NAME AS TS_NAME, ETSVIEW.ENCRYPTIONALG
   AS ENCRYPTION_ALG FROM V$TABLESPACE TSVIEW , V$ENCRYPTED_TABLESPACES
   ETSVIEW
   WHERE TSVIEW.TS# = ETSVIEW.TS#
   AND ETSVIEW.ENCRYPTEDTS = 'YES'
   AND ETSVIEW.ENCRYPTIONALG = 'AES128';
   ```

   Output similar to the following appears:

   ```
   CON_ID  TS_NAME           ENCRYPTION_ALG
   ------  ----------------  ------------------------
        1  ENCRYPTED_TS_CDB  AES128
   ```

2. Re-encrypt each tablepace that is listed in the output.

   For example, to migrate the `ENCRYPTED_TS_CDB` tablespace to use the latest supported algorithm, which is `AES256`:

   ```
   ALTER TABLESPACE ENCRYPTED_TS_CDB ENCRYPTION REKEY;
   ```

3. To verify the re-encryption:

```
SELECT C.NAME AS PDB_NAME, T.NAME AS TBS_NAME, E.ENCRYPTIONALG AS ALG,
E.CIPHERMODE AS "MODE", E.STATUS FROM V$TABLESPACE T,
V$ENCRYPTED_TABLESPACES E, V$CONTAINERS C
WHERE E.TS# = T.TS# AND E.CON_ID = T.CON_ID AND E.CON_ID = C.CON_ID
ORDER BY E.CON_ID, T.NAME;
```

6.11 Creating an Encrypted Database Using DBCA

You can use DBCA to create an encrypted database in both single instance multitenant and Oracle Data Guard environments.

6.11.1 Using DBCA to Create an Encrypted Database

Before you run DBCA to create an encrypted database, you must create the `WALLET_ROOT` directory.

1. Log into the host where DBCA will create the database with the united mode PDBs.

   DBCA cannot create PDBs in isolated mode.

2. Create the `WALLET_ROOT` directory.

   For example:

   ```
   mkdir -pvm700 /etc/ORACLE/KEYSTORES/finance
   ```

   When you run DBCA in the next step, DBCA will create the *`WALLET_ROOT`*`/tde` sub-directory.

3. Run the `dbca -createDatabase` command to create the encrypted database, and include the appropriate TDE-related parameters.

   For example:

   ```
   -configureTDE TRUE
       -tdeWalletRoot /etc/ORACLE/KEYSTORES/finance
       -tdeWalletLoginType LOCAL_AUTO_LOGIN
       -TdeWalletPassword tde_password
       -encryptPDBTablespaces ALL
   ```

   If you omit the following values, then DBCA uses their defaults, as follows:

   - `tdeAlgorithm`: The default is AES256 with XTS cipher mode.
   - `encryptTablespaces`: Only the `USERS` tablespace is encrypted in the `CDB$ROOT`.
   - `encryptPDBTablespaces`: The `SYSTEM`, `SYSAUX` and `USERS` tablespaces are encrypted. The `TEMP` and `UNDO` tablespaces remain unencrypted. The `ALL` setting encrypts all tablespaces in the PDB.

   If you add `-initParameter TABLESPACE_ENCRYPTION = AUTO_ENABLE`, then all tablespaces in `CDB$ROOT` and PDB are encrypted. This directive is incompatible with setting `-encryptPDBTablespaces` and `-encryptTablespaces`.

Related Topics

- *Oracle Multitenant Administrator's Guide*

6.11.2 Using DBCA to Create an Oracle Data Guard Standby Database from an Encrypted Primary Database

Before you run DBCA to create the encrypted database in an Oracle Data Guard environment, you must copy the wallet from the primary to the standby database, and then create the `WALLET_ROOT` directory.

1. Log into the server where DBCA will create the standby database from an encrypted primary database.

   DBCA cannot create a standby database from an encrypted primary database if the `REDO_TRANSPORT_USER` initialization parameter is set to a common user.

2. Copy the TDE wallet from the primary host to the standby host.

   When you run DBCA, you will specify this wallet location when you set the `primaryDBTdeWallet` parameter.

3. Create the `WALLET_ROOT` directory.

   For example:

   ```
   mkdir -pvm700 /etc/ORACLE/KEYSTORES/finance
   ```

   When you run DBCA in the next step, DBCA will create the *`WALLET_ROOT`*`/tde` sub-directory.

4. Run the `dbca -createDuplicateDB` command to create the encrypted standby database, and include the appropriate TDE-related parameters.

   For example:

   ```
   dbca -silent -createDuplicateDB -createAsStandby
     -configureTDE TRUE
       -tdeWalletRoot /etc/ORACLE/KEYSTORES/finance
       -primaryDBTdeWallet /tmp
       -tdeWalletLoginType LOCAL_AUTO_LOGIN
       -sourceTdeWalletPassword tde_password
       -TdeWalletPassword tde_password
   ```

Related Topics

- *Oracle Multitenant Administrator's Guide*

6.11.3 Best Practice after DBCA Creates an Encrypted Database

After DBCA has created an encrypted stand-alone or Oracle Data Guard primary and standby database, you can implement Transparent Data Encryption (TDE) best practices.

1. Add the wallet password to another (local) auto-open wallet so that the wallet password on the SQL*Plus command line can be replaced by `EXTERNAL STORE`.

This helps with separation of duties and automation. In Oracle Data Guard, you should run both commands on the primary and standby databases.

```
$ mkdir -pvm700 WALLET_ROOT/tde_seps

SQL> ADMINISTER KEY MANAGEMENT ADD SECRET 'wallet_password'
FOR CLIENT 'TDE_WALLET' TO [LOCAL] AUTO-LOGIN KEYSTORE 'wallet_root/
tde_seps';
```

2. Add a tag to the TDE master key for the PDB.

   When DBCA creates a TDE master key for the PDB, it does not add a tag to this key. The following `ADMINISTER KEY MANAGEMENT` command adds the tag to the PDB. The tag consists of the string *`pdb_name date time`* `(IN UTC)`. The date and time is derived from the time when the key was first activated, not the current time. Run this command in the primary PDB.

   ```
   SELECT ' ADMINISTER KEY MANAGEMENT
   SET TAG '''||SYS_CONTEXT('USERENV', 'CON_NAME')||' '||TO_CHAR
   (SYS_EXTRACT_UTC (ACTIVATION_TIME),
   'YYYY-MM-DD HH24:MI:SS"Z"')||''' FOR '''||KEY_ID||'''
   FORCE KEYSTORE IDENTIFIED BY EXTERNAL STORE WITH BACKUP;'
   AS "SET TAG COMMAND" FROM V$ENCRYPTION_KEYS;
   ```

   For Oracle Data Guard, copy the primary wallet (with the tagged PDB key) to the standby side. If you are using local auto-login wallets, then delete and recreate the local auto-open wallet on the standby database.

3. Encrypt the data dictionary.

   This is a Data Guard transaction and as such, it is automatically applied to the standby database. As a user with the `SYSKM` administrative privilege, perform the following `ALTER DATABASE DICTIONARY` command:

   ```
   ALTER DATABASE DICTIONARY ENCRYPT CREDENTIALS CONTAINER = CURRENT;
   ```

4. Optionally, you can isolate the encrypted PDB with a single command.

   a. From within the primary PDB:

      ```
      ADMINISTER KEY MANAGEMENT FORCE ISOLATE KEYSTORE IDENTIFIED BY
      "new_pdb_wallet_password"
      FROM ROOT KEYSTORE FORCE KEYSTORE
      IDENTIFIED BY EXTERNAL STORE WITH BACKUP;
      ```

      You can run this command without knowing the root wallet password, and the root administrator will not know the new isolated wallet password. This command performs the following actions:

      - Creates the *`WALLET_ROOT`*`/`*`pdb_guid`*`/tde` directrory
      - Creates a wallet in the *`WALLET_ROOT`*`/`*`pdb_guid`*`/tde` directory
      - Moves the PDB keys from the root wallet to the PDB wallet
      - Sets the `TDE_CONFIGURATION` parameter for the PDB to `FILE`

Next, create an isolated (local) auto-open wallet; from within the primary PDB:

```
ADMINISTER KEY MANAGEMENT CREATE [LOCAL] AUTO_LOGIN KEYSTORE
FROM KEYSTORE IDENTIFIED BY "new_pdb_wallet_password";
```

b. From within the isolated PDB, create a directory and another local auto-open wallet that enables you to hide the isolated wallet password from the SQL*Plus command line.

```
SELECT ' HOST MKDIR -PVM700 '||B.VALUE||'/'||A.GUID||'/TDE_SEPS'
AS "CREATE DIRECTORY FOR IDENTIFIED BY EXTERNAL STORE (IBES)"
FROM V$PDBS A, V$PARAMETER B WHERE B.NAME = 'WALLET_ROOT';

SELECT 'ADMINISTER KEY MANAGEMENT ADD SECRET ''NEW-PDB-WALLET-PWD''
FOR CLIENT ''TDE_WALLET'' TO LOCAL AUTO_LOGIN KEYSTORE '''||
B.VALUE||'/'||A.GUID||'/TDE_SEPS'''
AS "CREATE ISOLATED WALLET FOR IBES"
FROM V$PDBS A, V$PARAMETER B WHERE B.NAME = 'WALLET_ROOT';
```

6.12 Encryption Conversions for Existing Databases

You can encrypt both offline and online databases.

6.12.1 About Encryption Conversions for Existing Databases

The encryption conversion of an entire database encrypts all tablespaces, including the Oracle-supplied `SYSTEM`, `SYSAUX`, `UNDO`, and `TEMP` tablespaces.

Note the following:

- If you are using Database Configuration Assistant (DBCA) to create or configure a database, then you can create a TDE wallet in the database as part of the creation or configuration process. When you drop a database by using DBCA, any TDE wallets that are in this database are also removed. **Important: Before you drop a database by using DBCA, and if it has any local TDE wallets, then back these wallets up to a secure location.** If the database has been migrated to use Oracle Key Vault, then be aware that its TDE encryption keys reside in the Oracle Key Vault server. It is the responsibility of the Oracle Key Vault administrator to back up Oracle Key Vault.
- To perform the encryption, you can use the offline and online functionality of the tablespace encryption conversions.
- You can encrypt any or all of the Oracle-supplied tablespaces, and in any order. The encryption of the Oracle-supplied tablespaces has no impact on the encryption of user-created tablespaces.
- When you encrypt the Oracle-supplied tablespaces, Oracle Database prevents the keystore from being closed.
- You cannot encrypt an existing temporary tablespace, but you can create an encrypted temporary tablespace, configure the database to use that new encrypted temporary tablespace as the default temporary tablespace, and then drop the old clear-text temporary tablespace.
- The `UNDO` and `TEMP` metadata that is generated from sensitive data in an encrypted tablespace is already automatically encrypted. Therefore, encrypting `UNDO` and `TEMP` is optional.

- The performance effect of encrypting all the tablespaces in a database depends on the workload and platform. Many modern CPUs provide built-in hardware acceleration, which results in a minimal performance impact.
- In a multitenant environment, you can encrypt any tablespaces in any pluggable databases (PDBs), including the Oracle-supplied tablespaces. However, the keystore in the CDB root must be open at all times so that a PDB can open its keystore. You can check the status of whether a keystore is open by querying the `STATUS` column of the `V$ENCRYPTION_WALLET` view

6.12.2 Encrypting an Existing Database with Offline Conversion

When you encrypt an existing database with offline conversion, for the Oracle-managed tablespaces, you do not specify an encryption algorithm.

1. Connect to the united mode CDB root or isolated mode PDB as a user who has been granted the `SYSDBA` administrative privilege.

   You must have the `SYSDBA` administrative privilege to work with the `SYSTEM` and `SYSAUX` tablespaces. Otherwise, connect with the `SYSKM` administrative privilege.

2. Mount the database.

   ```
   STARTUP MOUNT
   ```

3. As a user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege, open the keystore.

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

4. Run the `ALTER TABLESPACE` SQL statement to encrypt the `SYSTEM`, `SYSAUX`, and `UNDO` tablespaces. Do not encrypt the `SYSTEM` tablespace concurrently with the encryption of other tablespaces.

   For example, to encrypt the `SYSTEM` tablespace:

   ```
   ALTER TABLESPACE SYSTEM ENCRYPTION OFFLINE ENCRYPT;
   ```

5. Open the CDB root or the PDB.
   - For a CDB, to open in read/write mode, for example:

     ```
     ALTER DATABASE OPEN READ WRITE;
     ```

   - For a PDB:

     ```
     ALTER PLUGGABLE DATABASE pdb_name OPEN READ WRITE;
     ```

6. Run the `ALTER TABLESPACE` SQL statement to encrypt other user tablespaces.

   Alternatively, you can proceed to the next step and open the database first, and then perform the steps to encrypt an existing user-defined tablespace with offline conversion.

7. Open the CDB root or the PDB.
   - For a CDB:

     ```
     ALTER DATABASE OPEN;
     ```

   - For a PDB:

     ```
     ALTER PLUGGABLE DATABASE pdb_name OPEN;
     ```

After you have encrypted the tablespace, if you want to use a different encryption algorithm (change the TDE master encryption key) for the `SYSTEM`, `SYSAUX`, and `UNDO` tablespaces, then

you must use online conversion. In addition to AES128, supported encryption algorithms are AES192 and AES256, in addition to other algorithms such as ARIA and GOST.

> **Note:**
>
> Starting with Oracle Database 23ai, the Transparent Data Encryption (TDE) decryption libraries for the GOST and SEED algorithms are deprecated, and encryption to GOST and SEED are desupported. Starting with Oracle Database 23ai, the Transparent Data Encryption (TDE) encryption libraries for the GOST and SEED algorithms are desupported and removed. The GOST and SEED decryption libraries are deprecated. Both are removed on HP Itanium platforms.
> GOST 28147-89 has been deprecated by the Russian government, and SEED has been deprecated by the South Korean government. If you need South Korean government-approved TDE cryptography, then use ARIA instead. If you are using GOST 28147-89, then you must decrypt and encrypt with another supported TDE algorithm. The decryption algorithms for GOST 28147-89 and SEED are included with Oracle Database 23ai, but are deprecated, and the GOST encryption algorithm is desupported with Oracle Database 23ai. If you are using GOST or SEED for TDE encryption, then Oracle recommends that you perform an online rekey operation before upgrading to Oracle Database 23ai. However, with the exception of the HP Itanium platform, the GOST and SEED decryption libraries are available with Oracle Database 23ai, so you can also decrypt after upgrading.

Related Topics

- Encrypting an Existing User-Defined Tablespace with Offline Conversion
  To encrypt an existing tablespace with offline conversion, you can use the ALTER TABLESPACE SQL statement with the OFFLINE and ENCRYPT clauses.
- Changing the TDE Master Encryption Key for a Tablespace
  You can use the ENCRYPT and REKEY clauses of the ALTER TABLESPACE statement to encrypt a tablespace.
- Supported Encryption and Integrity Algorithms
  Oracle supports the AES, ARIA and DES algorithms.

6.12.3 Encrypting an Existing Database with Online Conversion

When you encrypt an existing database with online conversion, you do not specify an encryption algorithm.

The reason that you do not need to specify an encryption algorithm the first time you perform the encryption is that the tablespaces that you must use to encrypt the database are automatically encrypted with the database key. If you want to change the algorithm, then you can issue the ALTER TABLESPACE ENCRYPTION REKEY SQL statement after the initial encryption.

1. Perform the following tasks, which are performed when encrypting an existing tablespace with online conversion:
   a. Connect as a user who has been granted the SYSDBA administrative privilege.
   b. Ensure that the COMPATIBLE parameter is set to 12.2.0.0 or later.
   c. Ensure that the database is open in read-write mode.
   d. Ensure that you have enough auxiliary space to complete the encryption.

e. Back up the tablespaces that you must encrypt.

f. Open the keystore.

2. Run the `ALTER TABLESPACE` SQL statement to encrypt the `SYSTEM`, `SYSAUX`, and `UNDO` tablespaces. Do not specify an algorithm, and do not encrypt the `SYSTEM` tablespace concurrently with the encryption of other tablespaces.

   For example, to encrypt the `SYSTEM` tablespace:

   ```
   ALTER TABLESPACE SYSTEM ENCRYPTION ONLINE ENCRYPT
   FILE_NAME_CONVERT=('system01.dbf','system01_enc.dbf');
   ```

3. Create a temporary tablespace.

   a. Create a new (encrypted) `TEMP` tablespace with the identical characteristics as the original `TEMP` tablespace, as follows:

   ```
   SELECT ' CREATE TEMPORARY TABLESPACE '
   ||tablespace_name||'_ENC tempfile '''
   ||substr(file_name,1,length(file_name)-4)
   ||'_enc.dbf'' size '
   ||bytes||' ENCRYPTION USING ''AES256'' ENCRYPT;'
   AS " Create new encrypted TEMP tablespace"
   FROM DBA_TEMP_FILES;
   ```

   Alternatively, you could extract the original parameters by running `DBMS_METADATA.GET_DDL`. If you omit the `USING` *`algorithm`* clause, then Oracle Database applies the default algorithm.

   b. Run the command (that is, the output of the `SELECT` statement).

   c. Change the database default temporary tablespace to the new encrypted `TEMP` tablespace.

   ```
   ALTER DATABASE DEFAULT TEMPORARY TABLESPACE temp_enc;
   ```

   d. Drop the original `TEMP` tablespace.

   ```
   DROP TABLESPACE TEMP INCLUDING CONTENTS AND DATAFILES;
   ```

   Optionally, you can rename `temp_enc` to `TEMP`.

Related Topics

- Encrypting an Existing Tablespace with Online Conversion
  To encrypt an existing tablespace with online conversion, use `ALTER TABLESPACE` with the `ONLINE` and `ENCRYPT` clauses.
- Changing the TDE Master Encryption Key for a Tablespace
  You can use the `ENCRYPT` and `REKEY` clauses of the `ALTER TABLESPACE` statement to encrypt a tablespace.

7

Managing the Keystore and the Master Encryption Key

You can modify settings for the keystore and TDE master encryption key, and store Oracle Database and store Oracle GoldenGate secrets in a keystore.

7.1 Managing the Keystore

You can perform maintenance activities on keystores such as changing passwords, and backing up, merging, and moving keystores.

7.1.1 Performing Operations That Require a Keystore Password

Many `ADMINISTER KEY MANAGEMENT` operations require access to a keystore password, for both TDE wallets and external keystores.

In some cases, a keystore depends on an auto-login TDE wallet before the operation can succeed. Auto-login TDE wallets open automatically when they are configured and a key is requested. They are generally used for operations where the TDE wallet could be closed but a database operation needs a key (for example, after the database is restarted). Because the auto-login TDE wallet opens automatically, it can be retrieved to perform a database operation without manual intervention. However, some keystore operations that require the keystore password cannot be performed when the auto-login keystore is open. The auto-login TDE wallet must be closed and the password-protected keystore must be opened for the keystore operations that require a password.

In a multitenant environment, the re-opening of keystores affects other PDBs. For example, an auto-login TDE wallet in the root must be accessible by the PDBs in the CDB for this root.

You can temporarily open the TDE wallet by including the `FORCE KEYSTORE` clause in the `ADMINISTER KEY MANAGEMENT` statement when you perform the following operations: rotating a TDE wallet password; creating, using, rekeying, tagging, importing, exporting, migrating, or reverse migrating encryption keys; opening or backing up TDE wallet; adding, updating, or deleting secret TDE wallets. In a multitenant environment, if no TDE wallet is open in the root, then `FORCE KEYSTORE` opens the password-protected TDE wallet in the root.

7.1.2 Configuring Auto-Open Connections into External Key Managers

An external key manager can be configured to use the auto-login capability.

7.1.2.1 About Auto-Open Connections into External Key Managers

An auto-open connection into an external key manager stores the external keystore credentials in an auto-login keystore.

You can configure a connection to an external key manager so that the database can open the keystore without prompting for the keystore password. This configuration is essential in Oracle Real Application Clusters (Oracle RAC) environments, and is highly recommended for Oracle Data Guard standby databases. Be aware that this configuration reduces the security of the

system as a whole. However, this configuration does support unmanned or automated operations, and is useful in deployments where TDE-enabled databases that are enrolled into an external keystore for key management can start automatically.

Be aware that running the query `SELECT * FROM V$ENCRYPTION_WALLET` will automatically open an auto-login external keystore. For example, suppose you have an auto-login external keystore configured. If you close the keystore and query the `V$ENCRYPTION_WALLET` view, then the output will indicate that a keystore is open. This is because `V$ENCRYPTION_WALLET` opened up the auto-login external keystore and then displayed the status of the auto-login keystore.

To enable the auto-login capability for an external keystore, you must store the external keystore's credentials in an auto-login wallet.

When you use the `ADMINISTER KEY MANAGEMENT` statement, there are conceptually two sets of commands that act on client secrets:

- `ADMINISTER KEY MANAGEMENT` commands that act on the wallet that is currently in use (in other words, a wallet that contains an active TDE master encryption key).
- `ADMINISTER KEY MANAGEMENT` commands that act on a wallet that is not currently being used to hold the active TDE master encryption key. Oracle recommends that you use this approach when you configure an auto-login external keystore.

7.1.2.2 Configuring an Auto-Open Connection into an External Key Manager

To configure the auto-open connection, you must use the `ADMINISTER KEY MANAGEMENT` statement to add or update a client secret to authenticate to the external key manager.

Before you begin this procedure, ensure that you have configured the external keystore.

In this procedure, the wallet that is created does not contain any keys. It only holds the client secret. So, when you query the `V$ENCRYPTION_WALLET` dynamic view for this wallet, the `STATUS` column shows `OPEN_NO_MASTER_KEY` rather than `OPEN`, because the wallet only contains the client secret.

1. Reconfigure the `WALLET_ROOT` parameter in the `init.ora` file to include the location of the TDE wallet, if it is not already present.

   The TDE wallet location may already be present if you had previously migrated to using the external key manager.

   For example:

   ```
   WALLET_ROOT=/etc/ORACLE/WALLETS/orcl
   ```

2. Add or update the secret in the TDE wallet.

   The secret is the external keystore password and the client is the `OKV_PASSWORD`. `OKV_PASSWORD` is an Oracle-defined client name that is used to represent the external key manager password as a secret in the TDE wallet.

   For example:

   ```
   ADMINISTER KEY MANAGEMENT ADD SECRET 'external_key_manager_password'
   FOR CLIENT 'OKV_PASSWORD'
   TO LOCAL AUTO_LOGIN KEYSTORE TDE_wallet_location
   WITH BACKUP;
   ```

   In this example:

   - *`TDE_wallet_location`* is the location of the TDE wallet within the `WALLET_ROOT` location that you just defined in Step 1.

For the CDB root and for any PDB that is configured in united mode, the value to use for the *TDE_wallet_location* location is *WALLET_ROOT*/tde.

For any PDB that is configured in isolated mode, the value to use for the *TDE_wallet_location* location is *WALLET_ROOT*/*pdb_guid*/tde. When you are in the PDB, run the following query to find this GUID: `SELECT GUID FROM V$PDBS;`

- `LOCAL` creates a local auto-login wallet file, `cwallet.sso`, to hold the credentials for the external key manager. This wallet is tied to the host on which it was created.
  For an Oracle Real Application Clusters environment, omit the `LOCAL` keyword, because each Oracle RAC node has a different host name, yet they all use the same external key manager. If you configure a local auto-login wallet for the Oracle RAC instance, then only the first Oracle RAC node, where the `cwallet.sso` file was created, would be able to access the external key manager credentials. If you try to open the TDE wallet from another node instead of from that first node, there would be a problem auto-opening `cwallet.sso`, and so it would result in a failure to auto-open the auto-login external keystore. This restriction applies if you are using a shared location to hold the `cwallet.sso` file for the Oracle RAC cluster, because using `LOCAL` only works if you have a separate `cwallet.sso` file (containing the same credentials) on each node of the Oracle RAC environment.

At this stage, the next time a TDE operation runs, the external key manager auto-login TDE wallet opens automatically. An example of a TDE operation is to query the `V$ENCRYPTION_WALLET` view, for example:

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

7.1.3 Changing the Oracle Key Vault Password

To change the password of Oracle Key Vault, you use `okvutil`, which is part of the Oracle Key Vault endpoint software on the database host.

1. Log in to the database instance as a user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.
2. Close the external keystore.
   - Close the connection to the external key manager:

     ```
     ADMINISTER KEY MANAGEMENT SET KEYSTORE CLOSE
     IDENTIFIED BY Oracle_Key_Vault_password | EXTERNAL STORE CONTAINER = ALL;
     ```

     If the keystore was auto-opened by the database, then close the connection to Oracle Key Vault as follows:

     ```
     ADMINISTER KEY MANAGEMENT SET KEYSTORE CLOSE
     CONTAINER = ALL;
     ```
3. Change the Oracle Key Vault password.

   WALLET_ROOT/okv/bin/okvutil changepwd -t wallet -l *WALLET_ROOT*/okv/ssl
4. Open the external keystore.
   - For example, for Oracle Key Vault:

     ```
     ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN FORCE KEYSTORE
     IDENTIFIED BY new_Oracle_Key_Vault_pwd CONTAINER =ALL;
     ```
   - If the old Oracle Key Vault password was stored in a [local] auto-open wallet in the *WALLET_ROOT*/tde_seps directory, then update the password using the following syntax:

```
ADMINISTER KEY MANAGEMENT UPDATE SECRET 'new-Oracle_Key_Vault_password'
FOR CLIENT 'Oracle_Key_Vault_password" TO [LOCAL] AUTO_LOGIN KEYSTORE
'WALLET_ROOT/tde_seps;
```

Then you can open the connection to Oracle Key Vault as follows:

```
ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN
[FORCE KEYSTORE] IDENTIFIED BY EXTERNAL STORE CONTAINER = ALL;
```

7.1.4 Configuring an External Store for a Keystore Password

An external store for a keystore password stores the keystore password in a centrally accessed and managed location.

7.1.4.1 About Configuring an External Store for a Keystore Password

An external store for a keystore password allows you to easily remove that keystore password from the `ADMINISTER KEY MANAGEMENT` command line.

This feature implements separation of duties between database administrators and key administrators. It is also useful for situations in which you use automated tools to perform Transparent Data Encryption operations that require a password, when the scripts that run the automated tools include hard-coded password. To avoid hard-coding the password in a script, you can store this password in an external store on the database server. In a multitenant environment, different PDBs can make use of the external store.

In a multitenant environment, all PDBs in united mode use the hidden password of the root container. In isolated mode, each PDB can have its own keystore password in its own external store.

Related Topics

- Storing Oracle Database Secrets in Isolated Mode
  Secrets are data that support internal Oracle Database features that integrate external clients such as Oracle GoldenGate into the database.

7.1.4.2 Configuring the External Keystore Password Store with WALLET_ROOT

When you configure TDE by using the `WALLET_ROOT` parameter, the external keystore password store is auto-discovered in the *`WALLET_ROOT`*`/tde_seps` directory.

- Run the `ADMINISTER KEY MANAGEMENT` statement by using the following syntax:

```
ADMINISTER KEY MANAGEMENT ADD SECRET 'keystore_password'
FOR CLIENT 'TDE_WALLET|OKV_PASSWORD'
TO [LOCAL] AUTO_LOGIN KEYSTORE 'WALLET_ROOT/tde_seps';
```

Related Topics

- When to Use the EXTERNAL STORE Clause After Configuration
  After you configure the external store for a keystore password, you can use the `EXTERNAL_STORE` clause in the `ADMINISTER KEY MANAGEMENT` statement.

7.1.4.3 When to Use the EXTERNAL STORE Clause After Configuration

After you configure the external store for a keystore password, you can use the `EXTERNAL_STORE` clause in the `ADMINISTER KEY MANAGEMENT` statement.

You must use the `EXTERNAL STORE` clause in the `ADMINISTER KEY MANAGEMENT` statement for the following operations: opening, closing, backing up the keystore; adding, updating, or deleting a secret keystore; creating, using, rekeying, tagging, importing, exporting encryption keys.

For example:

```
ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN
IDENTIFIED BY EXTERNAL STORE;
```

You can change or delete external keystore passwords by using the `ADMINISTER KEY MANAGEMENT UPDATE CLIENT SECRET` statement or the `ADMINISTER KEY MANAGEMENT DELETE CLIENT SECRET` statement.

7.1.5 Backing Up Password-Protected TDE Wallets

When you back up a password-protected TDE wallet, you can create a backup identifier string to describe the backup type.

7.1.5.1 About Backing Up Password-Protected TDE Wallets

You must back up password-protected TDE wallets, as per the security policy and requirements of your site.

A backup of the TDE wallet contains all of the keys contained in the original TDE wallet. Oracle Database prefixes the backup TDE wallet with the creation time stamp (UTC). If you provide an identifier string, then this string is inserted between the time stamp and TDE wallet name.

After you complete the backup operation, the keys in the original TDE wallet are marked as "backed up". You can check the status of keys querying the `V$ENCRYPTION_WALLET` data dictionary view.

You cannot back up auto-login or local auto-login TDE wallets. No new keys can be added to them directly through the `ADMINISTER KEY MANAGEMENT` statement operations. The information in these TDE wallets is only read and hence there is no need for a backup.

You must include the `WITH BACKUP` clause in any `ADMINISTER KEY MANAGEMENT` statement that changes the wallet (for example, changing the wallet password, or setting the master encryption key).

7.1.5.2 Creating a Backup Identifier String for the Backup TDE Wallet

The backup file name of a software password wallet is derived from the name of the password-protected TDE wallet.

- To create a backup identifier string for a backup TDE wallet, use the `ADMINISTER KEY MANAGEMENT` SQL statement with the `BACKUP KEYSTORE` clause, with the following syntax:

  ewallet_*creation-time-stamp-in-UTC_user-defined-string*.p12

  When you create the backup identifier (*user_defined_string*), use the operating system file naming convention. For example, in UNIX systems, you may want to ensure that this setting does not have spaces.

  The following example shows the creation of a backup TDE wallet that uses a user-identified string, and how the resultant TDE wallet appears in the file system. This example includes the `FORCE KEYSTORE` clause in the event the auto-login TDE wallet is in use or the TDE wallet is closed.

```
ADMINISTER KEY MANAGEMENT BACKUP KEYSTORE USING 'Monthly-backup-2013-04'
FORCE KEYSTORE
IDENTIFIED BY TDE_wallet_password;
```

This version is for a scenario in which the password is stored in an external store:

```
ADMINISTER KEY MANAGEMENT BACKUP KEYSTORE USING 'Monthly-backup-2013-04'
FORCE KEYSTORE
IDENTIFIED BY EXTERNAL STORE;
```

Resultant TDE wallet file:

```
ewallet_2013041513244657_Monthly-backup-2013-04.p12
```

7.1.5.3 Backing Up a Password-Protected TDE Wallet

The `BACKUP KEYSTORE` clause of the `ADMINISTER KEY MANAGEMENT` statement backs up a password-protected TDE wallet.

- Back up the keystore by using the following syntax:

```
ADMINISTER KEY MANAGEMENT BACKUP KEYSTORE
[USING 'backup_identifier']
FORCE KEYSTORE]
IDENTIFIED BY [EXTERNAL STORE | TDE_wallet_password]
[TO 'keystore_location'];
```

In this specification:

- `USING` *backup_identifier* is an optional string that you can provide to identify the backup. Enclose this identifier in single quotation marks (`' '`). This identifier is appended to the named keystore file (for example, `ewallet_`*time-stamp*`_emp_key_backup.p12`).
- `FORCE KEYSTORE` temporarily opens the password-protected TDE wallet for this operation. You must open the TDE wallet for this operation.
- `IDENTIFIED BY` can be one of the following settings:
  - `EXTERNAL STORE` uses the keystore password stored in the external store to perform the keystore operation.
  - *TDE_wallet_password* is the password for the keystore.
- *keystore_location* is the path at which the backup keystore is stored. If you do not specify the *keystore_location*, then the backup is created in the same directory as the original keystore. Enclose this location in single quotation marks (`' '`).

The following example backs up a TDE wallet into another location.

```
ADMINISTER KEY MANAGEMENT BACKUP KEYSTORE
USING 'hr.emp_wallet'
FORCE KEYSTORE
IDENTIFIED BY TDE_wallet_password
TO '/etc/ORACLE/KEYSTORE/DB1/';

keystore altered.
```

In the following version, the password for the TDE wallet is external, so the `EXTERNAL STORE` clause is used. The TDE wallet is backed up into the same directory as the current TDE wallet.

```
ADMINISTER KEY MANAGEMENT BACKUP KEYSTORE
USING 'hr.emp_wallet'
FORCE KEYSTORE
IDENTIFIED BY EXTERNAL STORE;
```

After you run this statement, an `ewallet_identifier.p12` file (for example, `ewallet_time-stamp_hr.emp_wallet.p12`) appears in the keystore location.

7.1.6 How the V$ENCRYPTION_WALLET View Interprets Backup Operations

The `BACKUP` column of the `V$ENCRYPTION_WALLET` view indicates a how a copy of the keystore was created.

The column indicates if a copy of the keystore had been created with the `WITH BACKUP` clause of the `ADMINISTER KEY MANAGEMENT` statement or the `ADMINISTER KEY MANAGEMENT BACKUP KEYSTORE` statement.

When you modify a key or a secret, the modifications that you make do not exist in the previously backed-up copy, because you make a copy and then modify the key itself. Because there is no copy of the modification in the previous keystores, the `BACKUP` column is set to `NO`, even if the `BACKUP` had been set to `YES` previously. Hence, if the `BACKUP` column is `YES`, then after you perform an operation that requires a backup, such as adding a custom attribute tag, the `BACKUP` column value changes to `NO`.

7.1.7 Backups of the External Keystore

You cannot use Oracle Database to back up external keystores.

Uploading TDE wallets into Oracle Key Vault is another way of backing up the wallet and having it available immediately if the need arises (for example after accidental deletion of the wallet, or file corruption). If the database is not migrated to online key management with Oracle Key Vault, then it keeps relying on the TDE wallet, even if the wallet has been uploaded into Oracle Key Vault.

You can use the Oracle Key Vault `okvutil upload` and `okvutil download` commands to upload and download TDE wallets to and from Oracle Key Vault.

For example, to upload a TDE wallet to Oracle Key Vault:

```
$ okvutil upload -l "/etc/oracle/wallets" -t wallet -g "HRWallet"
Enter wallet password (<enter> for auto-login): password
Enter Oracle Key Vault endpoint password: Key_Vault_endpoint_password
```

This example shows how to download a TDE wallet from Oracle Key Vault:

```
$ okvutil download -l "/etc/oracle/wallets/orcl/" -t WALLET -g HRWallet
Enter new wallet password(<enter> for auto-login): Oracle_wallet_password
Confirm new wallet password: Oracle_wallet_password
Enter Oracle Key Vault endpoint password: Key_Vault_endpoint_password
```

Related Topics

- *Oracle Key Vault Administrator's Guide*

7.1.8 Merging TDE Wallets

You can merge TDE wallets in a variety of ways.

7.1.8.1 About Merging TDE Wallets

You can merge any combination of TDE wallets, but the merged keystore must be password-protected. It can have a password that is different from the constituent wallets.

To use the merged TDE wallet, you must explicitly open the merged TDE wallet after you create it, even if one of the constituent TDE wallets was already open before the merge.

Whether a common key from two source TDE wallets is added or overwritten to a merged TDE wallet depends on how you write the `ADMINISTER KEY MANAGEMENT` merge statement. For example, if you merge TDE wallet 1 and TDE wallet 2 to create TDE wallet 3, then the key in TDE wallet 1 is added to TDE wallet 3. If you merge TDE wallet 1 into TDE wallet 2, then the common key in TDE wallet 2 is not overwritten.

The `ADMINISTER KEY MANAGEMENT` merge statement has no bearing on the configured TDE wallet that is in use. However, the merged TDE wallet can be used as the new configured database TDE wallet if you want. Remember that you must reopen the TDE wallet if you are using the newly created TDE wallet as the TDE wallet for the database at the location configured by the `WALLET_ROOT` parameter.

Related Topics

- Migrating from a TDE Wallet to Oracle Key Vault
  You can migrate between password-protected TDE wallets and external keystores in Oracle Key Vault.

7.1.8.2 Merging One TDE Wallet into an Existing TDE Wallet

You can use the `ADMINISTER KEY MANAGEMENT` statement with the `MERGE KEYSTORE` clause to merge one TDE wallet into another existing TDE wallet.

- To perform this type of merge, use the following SQL statement:

```
ADMINISTER KEY MANAGEMENT MERGE KEYSTORE 'TDE_wallet1_location'
[IDENTIFIED BY TDE_wallet1_password]
INTO EXISTING KEYSTORE 'TDE_wallet2_location'
IDENTIFIED BY TDE_wallet2_password
[WITH BACKUP [USING 'backup_identifier]];
```

  In this specification:

  - *TDE_wallet1_location* is the directory location of the first TDE wallet, which will be left unchanged after the merge. Enclose this path in single quotation marks (' ').
  - The `IDENTIFIED BY` clause is required for the first TDE wallet if it is a password-protected TDE wallet. *TDE_wallet1_password* is the password for the first TDE wallet.
  - *TDE_wallet2_location* is the directory location of the second TDE wallet into which the first TDE wallet is to be merged. Enclose this path in single quotation marks (' ').
  - *TDE_wallet2_password* is the password for the second keystore.
  - `WITH BACKUP` creates a backup of the TDE wallet. Optionally, you can use the `USING` clause to add a brief description of the backup. Enclose this description in single quotation marks (' '). This identifier is appended to the named TDE wallet file (for

example, `ewallet_time_stamp_emp_key_backup.p12`, with *emp_key_backup* being the backup identifier). Follow the file naming conventions that your operating system uses.

The resultant TDE wallet after the merge operation is always a password-protected TDE wallet.

7.1.8.3 Merging Two TDE Wallets into a Third New TDE Wallet

You can merge two TDE wallets into a third new TDE wallet. The two existing source TDE wallets are not changed.

- Merge the TDE wallets by using the following syntax:

```
ADMINISTER KEY MANAGEMENT MERGE KEYSTORE 'TDE_wallet1_location'
[IDENTIFIED BY TDE_wallet1_password]
AND KEYSTORE 'TDE_wallet2_location'
[IDENTIFIED BY TDE_wallet2_password]
INTO NEW KEYSTORE 'TDE_wallet3_location'
IDENTIFIED BY TDE_wallet3_password;
```

  In this specification:

  - *TDE_wallet1_location* is the directory location of the first TDE wallet, which will be left unchanged after the merge. Enclose this path in single quotation marks (`' '`).
  - The `IDENTIFIED BY` clause is required for the first TDE wallet if it is a password-protected TDE wallet. *TDE_wallet1_password* is the current password for the first TDE wallet.
  - *TDE_wallet2_location* is the directory location of the second TDE wallet. Enclose this path in single quotation marks (`' '`).
  - The `IDENTIFIED BY` clause is required for the second TDE wallet if it is a password-protected TDE wallet. *TDE_wallet2_password* is the current password for the second TDE wallet.
  - *TDE_wallet3_location* specifies the directory location of the new, merged TDE wallet. Enclose this path in single quotation marks (`' '`). If there is already an existing TDE wallet at this location, the command exits with an error.
  - *TDE_wallet3_password* is the new password for the merged TDE wallet.

  The following example merges an auto-login TDE wallet with a password-protected TDE wallet to create a merged password-protected TDE wallet at a new location:

```
ADMINISTER KEY MANAGEMENT MERGE KEYSTORE '/etc/ORACLE/KEYSTORE/DB1'
AND KEYSTORE '/etc/ORACLE/KEYSTORE/DB2'
IDENTIFIED BY existing_password_for_keystore_2
INTO NEW KEYSTORE '/etc/ORACLE/KEYSTORE/DB3'
IDENTIFIED BY new_password_for_keystore_3;

keystore altered.
```

7.1.8.4 Merging an Auto-Login TDE Wallet into an Existing Password-Protected TDE Wallet

You can merge an auto-login TDE wallet into an existing password-protected TDE wallet.

- Use the `ADMINISTER KEY MANAGEMENT MERGE KEYSTORE` SQL statement to merge an auto-login TDE wallet into an existing password-protected TDE wallet.

The following example shows how to merge an auto-login TDE wallet into a password-protected TDE wallet. It also creates a backup of the second TDE wallet before creating the merged TDE wallet.

```
ADMINISTER KEY MANAGEMENT MERGE KEYSTORE '/etc/ORACLE/KEYSTORE/DB1'
INTO EXISTING KEYSTORE '/etc/ORACLE/KEYSTORE/DB2'
IDENTIFIED BY keystore_password WITH BACKUP;
```

In this specification:

- `MERGE KEYSTORE` must specify the auto-login TDE wallet.
- `EXISTING KEYSTORE` refers to the password TDE wallet.

7.1.8.5 Reversing a TDE Wallet Merge Operation

You cannot directly reverse a TDE wallet merge operation.

When you merge a TDE wallet into an existing TDE wallet (rather than creating a new one), you must include the `WITH BACKUP` clause in the `ADMINISTER KEY MANAGEMENT` statement to create a backup of this existing TDE wallet. Later on, if you decide that you must reverse the merge, you can replace the merged TDE wallet with the one that you backed up. In other words, suppose you want merge TDE wallet A into TDE wallet B. By using the `WITH BACKUP` clause, you create a backup for TDE wallet B before the merge operation begins. (The original TDE wallet A is still intact.) To reverse the merge operation, revert to the backup that you made of TDE wallet B.

- Use the `ADMINISTER KEY MANAGEMENT MERGE KEYSTORE` SQL statement to perform merge operations.
  - For example, to perform a merge operation into an existing TDE wallet:

    ```
    ADMINISTER KEY MANAGEMENT MERGE KEYSTORE '/etc/ORACLE/KEYSTORE/DB1'
    INTO EXISTING KEYSTORE '/etc/ORACLE/KEYSTORE/DB2'
    IDENTIFIED BY password WITH BACKUP USING "merge1";
    ```

    Replace the new TDE wallet with the backup TDE wallet, which in this case would be named `ewallet_time-stamp_merge1.p12`.
  - To merge an auto-login TDE wallet into a password-based TDE wallet, use the `ADMINISTER KEY MANAGEMENT MERGE KEYSTORE` SQL statement.

7.1.9 Moving a TDE Wallet to a New Location

You move a TDE wallet to a new location after you have updated the `WALLET_ROOT` parameter.

If you are using Oracle Key Vault, then you can configure a TDE direct connection where Key Vault directly manages the master encryption keys. In this case, you will never need to manually move the TDE wallet to a new location.

1. Connect to the united mode CDB root or isolated mode PDB as a user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.
2. Back up the TDE wallet.

   For example:

   ```
   ADMINISTER KEY MANAGEMENT BACKUP KEYSTORE
   USING 'hr.emp_keystore'
   FORCE KEYSTORE
   IDENTIFIED BY
   TDE_wallet_password TO '/etc/ORACLE/KEYSTORE/DB1/';
   ```

3. Close the TDE wallet.

   Examples of ways that you can close the TDE wallet are as follows.

   For an auto-login TDE wallet:

   ```
   ADMINISTER KEY MANAGEMENT SET KEYSTORE CLOSE;
   ```

   For a password-protected TDE wallet:

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

   For a TDE wallet for which the password is stored externally:

   ```
   ADMINISTER KEY MANAGEMENT SET KEYSTORE CLOSE
   IDENTIFIED BY EXTERNAL STORE;
   ```

4. Exit the database session.

   For example, if you are logged in to SQL*Plus:

   ```
   EXIT
   ```

5. In the `init.ora` file for the database instance, update the `WALLET_ROOT` parameter to point to the new location where you want to move the TDE wallet.
6. Use the operating system move command (such as `mv`) to move the TDE wallet with all of its keys to the new directory location.

Related Topics

- *Oracle Key Vault Administrator's Guide*

7.1.10 Moving a TDE Wallet Out of Automatic Storage Management

You can use the `ADMINISTER KEY MANAGEMENT` statement to move a TDE wallet out Automatic Storage Management.

1. Connect to the united mode CDB root or isolated mode PDB as a user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.
2. Initialize a target TDE wallet on the file system by using the following syntax:

   ```
   ADMINISTER KEY MANAGEMENT CREATE KEYSTORE target_TDE_wallet_path
   IDENTIFIED BY target_TDE_wallet_password;
   ```

   In this specification:

   - *target_TDE_wallet_path* is the directory path to the target TDE wallet on the file system.
   - *target_TDE_wallet_password* is a password that you create for the TDE wallet.

   For example:

   ```
   ADMINISTER KEY MANAGEMENT CREATE KEYSTORE '/etc/ORACLE/KEYSTORE/DB1/' IDENTIFIED BY
   "target_TDE_wallet_password";
   ```

3. Copy the TDE wallet from ASM to the target TDE wallet that you just created.

   This step requires that you merge the TDE wallet from ASM to the file system, as follows:

   ```
   ADMINISTER KEY MANAGEMENT MERGE KEYSTORE source_TDE_wallet_path
   IDENTIFIED BY source_TDE_wallet_password
   INTO EXISTING KEYSTORE target_TDE_wallet_path
   ```

```
IDENTIFIED BY target_TDE_wallet_password
WITH BACKUP USING backupIdentifier;
```

In this specification:

- *source_TDE_wallet_path* is the directory path to the source TDE wallet.
- *source_TDE_wallet_password* is th source TDE wallet password.
- *target_TDE_wallet_path* is the path to the target TDE wallet.
- *target_TDE_wallet_password* is the target TDE wallet password.
- *backupIdentifier* is the backup identifier to be added to the backup file name.

For example:

```
ADMINISTER KEY MANAGEMENT MERGE KEYSTORE '+DATAFILE'
IDENTIFIED BY "source_TDE_wallet_password"
INTO EXISTING KEYSTORE '/etc/ORACLE/KEYSTORE/DB1/'
IDENTIFIED BY "target_TDE_wallet_password"
WITH BACKUP USING "bkup";
```

7.1.11 Migrating from a TDE Wallet to Oracle Key Vault

You can migrate between password-protected TDE wallets and external keystores in Oracle Key Vault.

7.1.11.1 Migrating from a Password-Protected TDE Wallet to an External Keystore

You can migrate from a password-protected TDE wallet to an external keystore.

7.1.11.1.1 Step 1: Convert the TDE Wallet to Open with the External Keystore

Some Oracle tools require access to the old TDE wallet to encrypt or decrypt data that was exported or backed up using the TDE wallet.

Examples of these tools are Oracle Data Pump and Oracle Recovery Manager.

- Use the `ADMINISTER KEY MANAGEMENT` SQL statement to convert a TDE wallet to open with an external keystore.
  - To set the TDE wallet password as that of the external keystore, use the following syntax:

    ```
    ADMINISTER KEY MANAGEMENT ALTER KEYSTORE PASSWORD
    FORCE KEYSTORE
    IDENTIFIED BY TDE_wallet_password
    SET "external_key_manager_password" WITH BACKUP
    [USING 'backup_identifier'];
    ```

    In this specification:

    * *TDE_wallet_password* is the same password that you used when creating the TDE wallet.
    * *external_key_manager_password* is the new TDE wallet password which is the same as the password of the external keystore.
    * `WITH BACKUP` creates a backup of the TDE wallet. Optionally, you can use the `USING` clause to add a brief description of the backup. Enclose this description in single quotation marks (`' '`). This identifier is appended to the named TDE wallet file (for example, `ewallet_`*time-stamp_emp_key_backup*`.p12`, with *emp_key_backup*

being the backup identifier). Follow the file naming conventions that your operating system uses.

– To create an auto-login TDE wallet for a TDE wallet, use the following syntax:

```
ADMINISTER KEY MANAGEMENT CREATE [LOCAL] AUTO_LOGIN KEYSTORE
FROM KEYSTORE 'keystore_location'
IDENTIFIED BY TDE_wallet_password;
```

In this specification:

* `LOCAL` enables you to create a local auto-login TDE wallet. Otherwise, omit this clause if you want the TDE wallet to be accessible by other computers.
* *`TDE_wallet_location`* is the path to the TDE wallet directory location of the wallet that is configured in the `sqlnet.ora` file.
* *`TDE_wallet_password`* is the existing password of the configured TDE wallet.

7.1.11.1.2 Step 2: Configure the External Keystore Type

You can use the `ALTER SYSTEM` statement to configure the external keystore type.

For the TDE wallet to open with the external keystore, either the TDE wallet must have the same password as the external keystore, or alternatively, you can create an auto-login TDE wallet for the TDE wallet.

1. Connect to the united mode CDB root or isolated mode PDB as a user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.
2. Set the `TDE_CONFIGURATION` dynamic initialization parameter.

   This example migrates the database from a TDE wallet to Oracle Key Vault.

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

7.1.11.1.3 Step 3: Perform the External Keystore Migration

You can use the `ADMINISTER KEY MANAGEMENT` SQL statement to perform an external keystore migration.

To migrate from the TDE wallet to external keystore, you must use the `MIGRATE USING` *`external_key_manager_password`* clause in the `ADMINISTER KEY MANAGEMENT SET KEY` SQL statement to decrypt the existing TDE table keys and tablespace encryption keys with the TDE master encryption key in the TDE wallet and then reencrypt them with the newly created TDE master encryption key in the external keystore. After you complete the migration, you do not need to restart the database, nor do you need to manually re-open the external keystore. The migration process automatically reloads the keystore keys in memory.

- Migrate the external keystores by using the following syntax:

  ```
  ADMINISTER KEY MANAGEMENT SET ENCRYPTION KEY
  IDENTIFIED BY "external_key_manager_password"
  MIGRATE USING TDE_wallet_password
  [WITH BACKUP [USING 'backup_identifier']];
  ```

  In this specification:

  – *`external_key_manager_password`* is the password that was created when the external keystore was created. Enclose this setting in double quotation marks (`" "`).

- *TDE_wallet_password* is the same password that you used when you created the TDE wallet or that you have changed to (when converting a TDE wallet to open with an external keystore).
- USING enables you to add a brief description of the backup. Enclose this description in single quotation marks (' '). This identifier is appended to the named keystore file (for example, ewallet_*time-stamp_emp_key_backup*.p12, with *emp_key_backup* being the backup identifier). Follow the file naming conventions that your operating system uses.

7.1.11.2 Migrating from an External Keystore to a Password-Based TDE Wallet

You can migrate an external keystore to a TDE wallet.

7.1.11.2.1 About Migrating Back from an External Keystore

To switch from using an external keystore solution to a TDE wallet, you can use reverse migration of the TDE wallet.

After you complete the switch, keep the external keystore, in case earlier backup files rely on the TDE master encryption keys in the external key manager.

If you had originally migrated from the TDE wallet to the external keystore and reconfigured the TDE wallet, then you already have an existing TDE wallet with the same password as the external keystore password. Reverse migration configures this keystore to act as the new TDE wallet with a new password. If your existing TDE wallet is an auto-login TDE wallet and you have the password-based TDE wallet for this auto-login TDE wallet, then use the password-based TDE wallet. If the password-based TDE wallet is not available, then merge the auto-login TDE wallet into a newly created empty password-based TDE wallet, and use the newly created password-based TDE wallet.

If you do not have an existing TDE wallet, then you must specify a TDE wallet location using the WALLET_ROOT parameter in the init.ora file. When you perform the reverse migration, migrate to the previous TDE wallet so that you do not lose the keys.

Related Topics

- Merging TDE Wallets
  You can merge TDE wallets in a variety of ways.
- Migration of an Encrypted Database from a TDE Wallet to Oracle Key Vault or OCI KMS
  To switch from a TDE wallet to centralized key management with Oracle Key Vault or Oracle Cloud Infrastructure (OCI) Key Management Service (KMS), after you upload all current and retired TDE master keys you must migrate the database from the TDE wallet to Oracle Key Vault or OCI KMS.

7.1.11.2.2 Step 1: Configure the External Keystore Type

You can use the ALTER SYSTEM statement to configure the external keystore type.

1. Connect to the united mode CDB root or isolated mode PDB as a user who has been granted the ALTER SYSTEM privilege.
2. Set the TDE_CONFIGURATION dynamic initialization parameter to specify the keystore type.

   For example:

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

Setting KEYSTORE_CONFIGURATION to "OKV" indicates a configuration where the connection into Oracle Key Vault must be opened by providing the Oracle Key Vault password. To use an auto-open Oracle Key Vault configuration, you set KEYSTORE_CONFIGURATION to "OKV|FILE", where the Oracle Key Vault password is stored in an auto-open keystore in *WALLET_ROOT*/tde.

7.1.11.2.3 Step 2: Configure the Keystore for the Reverse Migration

The ADMINISTER KEY MANAGEMENT statement with the SET ENCRYPTION KEY and REVERSE MIGRATE clauses can be used to reverse the migration of a keystore.

1. Connect to the united mode CDB root or isolated mode PDB as a user who has been granted the ADMINISTER KEY MANAGEMENT or SYSKM privilege.
2. Reverse migrate the keystore by using the following syntax:

```
ADMINISTER KEY MANAGEMENT SET ENCRYPTION KEY
IDENTIFIED BY TDE_wallet_password
REVERSE MIGRATE USING "external_key_manager_password"
[WITH BACKUP [USING 'backup_identifier']];
```

   In this specification:

   - *TDE_wallet_password* is the password for the existing keystore or the new keystore.
   - *external_key_manager_password* is the password that was created when you first created the external keystore. If the pre-external TDE wallet is the new keystore, then you must ensure that it has the same password as the *external_key_manager_password* before issuing the reverse migration command. Enclose this setting in double quotation marks (" ").
   - WITH BACKUP creates a backup of the TDE wallet. Optionally, you can include the USING clause to add a brief description of the backup. Enclose this description in single quotation marks (' '). This identifier is appended to the named keystore file (for example, ewallet_*time-stamp_emp_key_backup*.p12, with *emp_key_backup* being the backup identifier). Follow the file naming conventions that your operating system uses.

   For example:

```
ADMINISTER KEY MANAGEMENT SET ENCRYPTION KEY
IDENTIFIED BY TDE_wallet_password
REVERSE MIGRATE USING "external_key_manager_password" WITH BACKUP;

keystore altered.
```

3. Optionally, change the keystore password.

   For example:

```
ADMINISTER KEY MANAGEMENT ALTER KEYSTORE PASSWORD
IDENTIFIED BY old_TDE_wallet_password
SET new_TDE_wallet_password
WITH BACKUP USING 'before_password_was_changed';
```

7.1.11.2.4 Step 3: Configure the External Keystore to Open with the TDE Wallet

After you complete the migration, the migration process automatically reloads the keystore keys in memory.

You do not need to restart the database, nor do you need to manually re-open the TDE wallet.

The external keystore may still be required after reverse migration because the old keys are likely to have been used for Oracle Data Pump Export and Oracle Recovery Manager (Oracle RMAN)-encrypted backups. You should add the external keystore credentials to the keystore so that the HSM can be opened with the TDE wallet.

Related Topics

- Configuring Auto-Open Connections into External Key Managers
  An external key manager can be configured to use the auto-login capability.

7.1.11.3 Keystore Order After a Migration

After you perform a migration, keystores can be either primary or secondary in their order.

The `WALLET_ORDER` column of the `V$ENCRYPTION_WALLET` dynamic view describes whether a TDE wallet is primary (that is, it holds the current TDE master encryption key) or if it is secondary (it holds the previous TDE master encryption key). The `WRL_TYPE` column describes the type of locator for the TDE wallet (for example, `FILE` for the `sqlnet.ora` file). The `WALLET_ORDER` column shows `SINGLE` if two TDE wallets are not configured together and no migration was ever performed previously.

Table 7-1 describes how the keystore order works after you perform a migration.

Table 7-1 Keystore Order After a Migration

Type of Migration Done	WRL_TYPE	WALLET_ORDER	Description
Migration of TDE wallet to external keystore	`OKV` `FILE`	`PRIMARY` `SECONDARY`	Both the external and TDE wallet are configured. The TDE master encryption key can be either in Oracle Key Vault or the TDE wallet. The TDE master encryption key is first searched in Oracle Key Vault. If the TDE master encryption key is not in the primary keystore (Oracle Key Vault), then it will be searched for in the TDE wallet. All of the new TDE master encryption keys will be created in the primary keystore (in this case, Oracle Key Vault).
Reverse migration from Oracle Key Vault (`HSM`) to TDE wallet	`FILE` `HSM`	`PRIMARY` `SECONDARY`	Both the external and TDE wallet are configured. The TDE master encryption key can be either in the external keystore or the TDE wallet. The TDE master encryption key is first searched for in the TDE wallet. If the TDE master encryption key is not present in the primary (that is, software) TDE wallet, then it will be searched for in the HSM's external keystore. All of the new TDE master encryption keys will be created in the primary keystore (in this case, the TDE wallet).

7.1.12 Migration of Keystores to and from Oracle Key Vault

You can use Oracle Key Vault to migrate both TDE wallets and external keystores to and from Oracle Key Vault.

This enables you to manage the keystores centrally, and then share the keystores as necessary with other TDE-enabled databases in your enterprise.

Oracle Key Vault enables you to upload a keystore to a container called a virtual wallet, and then create a new virtual wallet from the contents of previously uploaded keystore. For example, suppose you previously uploaded a keystore that contains 5 keys. You can create a new virtual wallet that consists of only 3 of these keys. You then can download this keystore to another TDE-enabled database. This process does not modify the original keystore.

In addition to Oracle keystores, Oracle Key Vault enables you to securely share other security objects, such as credential files and Java keystores, across the enterprise. It prevents the loss of keys and keystores due to forgotten passwords or accidentally deleted keystores. You can use Oracle Key Vault with products other than TDE: Oracle Real Application Security, Oracle Active Data Guard, and Oracle GoldenGate. Oracle Key Vault facilitates the movement of encrypted data using Oracle Data Pump and Oracle Transportable Tablespaces.

Related Topics

- *Oracle Key Vault Administrator's Guide*

7.1.13 Configuring Keystores for Automatic Storage Management

You can store a TDE wallet on an Automatic Storage Management (ASM) disk group.

7.1.13.1 About Configuring Keystores for Automatic Storage Management

You can configure a TDE wallet for Automatic Storage Management (ASM) for a standalone database or a multitenant environment. The `WALLET_ROOT` location can be compliant or non-compliant with Oracle Managed File (OMF) systems.

You should use the `WALLET_ROOT` and `TDE_CONFIGURATION` initialization parameters to configure the TDE wallet location in an ASM system. The `TDE_CONFIGURATION` parameter must be set with the attribute `KEYSTORE_CONFIGURATION=FILE` in order for the `WALLET_ROOT` parameter to work. Note that starting with Oracle Database release 19c, the `ENCRYPTION_WALLET_LOCATION`, set in the `sqlnet.ora` file, is deprecated in favor of `WALLET_ROOT` and `TDE_CONFIGURATION`.

To perform the configuration, you must specify a `+` sign, followed by the ASM disk group and path where the TDE wallet will be located. For example:

```
WALLET_ROOT=+disk_group/path
```

Note the following:

- When you open a local TDE wallet, it opens only on the ASM node on which it was created.
- When you designate the path for the `WALLET_ROOT` for databases in standalone or multitenant environments, or environments where the `WALLET_ROOT` location either complies or does not comply with the Oracle Managed File (OMF) directory naming convention, be aware that this path must follow certain conventions so that the database can automate the creation of the directory components of the TDE wallet locations for you. Otherwise, you must manually create the directories under the `WALLET_ROOT` location.
- If you must move or merge TDE wallets between a regular file system and an ASM file system, then you can use the same TDE wallet merge statements that are used to merge TDE wallets.
- To run commands to manage TDE wallets in an ASM environment, you can use the `ASMCMD` utility.

Related Topics

- Merging TDE Wallets
  You can merge TDE wallets in a variety of ways.
- *Oracle Automatic Storage Management Administrator's Guide*

7.1.13.2 Configuring a Keystore to Point to an ASM Location

You can set `WALLET_ROOT` to point to an ASM directory within which the TDE wallet of the CDB root (which all united mode PDBs share) and the TDE wallets of all isolated mode PDBs are located.

1. Ensure that the `KEYSTORE_CONFIGURATION` attribute of the `TDE_CONFIGURATION` dynamic initialization parameter is set to `FILE`.

   For a CDB, set `TDE_CONFIGURATION` in the CDB root; for an isolated PDB, set it in the PDB.

   For example, in SQL*Plus:

   ```
   ALTER SYSTEM SET TDE_CONFIGURATION="KEYSTORE_CONFIGURATION=FILE";
   ```

2. Set the `WALLET_ROOT` static initialization parameter to the ASM disk group location followed by the `DB_UNIQUE_NAME` initialization parameter value.

   The inclusion of the value of `DB_UNIQUE_NAME` is necessary to allow the database server to automate the creation of the necessary directories under this location.

   You must not use `OMF` as a directory component of the `WALLET_ROOT` location (unlike in the standalone database configuration section).

   For example:

   ```
   WALLET_ROOT=+disk_group_name/db_unique_name
   ```

   This setting locates the TDE wallet that is used by the root and by all of the united mode PDBs in the *`WALLET_ROOT`*`/`*`db_unique_name`*`/tde` directory (that is, in `+`*`disk_group_name`*`/`*`db_unique_name`*`/tde`).

   This setting locates the TDE wallet which is used by each isolated mode PDB in the *`WALLET_ROOT`*`/`*`db_unique_name`*`/`*`pdb_guid`*`/tde` directory (that is, in `+`*`disk_group_name`*`/`*`db_unique_name`*`/`*`pdb_guid`*`/tde`).

7.1.13.3 Configuring a Keystore to Point to an ASM Location When the WALLET_ROOT Location Does Not Follow OMF Guidelines

If the chosen `WALLET_ROOT` location does not comply with the Oracle Managed File (OMF) guidelines, then the Oracle database cannot perform automation of the directory creation.

In this case, you must use the `ALTER DISKGROUP` command to manually create the necessary directories under the `WALLET_ROOT` location. You must use the `ALTER DISKGROUP ... ADD DIRECTORY` statement to manually create the necessary directories, because no automation of the directory creation is possible when the `WALLET_ROOT` parameter is not using an OMF-compliant value.

1. Connect to the united mode CDB root or isolated mode PDB using the `SYSASM` administrative privilege.
2. Ensure that the `KEYSTORE_CONFIGURATION` attribute of the `TDE_CONFIGURATION` dynamic initialization parameter is set to `FILE`.

For example:

```
ALTER SYSTEM SET TDE_CONFIGURATION="KEYSTORE_CONFIGURATION=FILE";
```

3. In the `init.ora` file, set the `WALLET_ROOT` static initialization parameter to the ASM disk group location.

   For example, the following path after *`disk_group_name`* contains no uppercase OMF directory elements:

   ```
   WALLET_ROOT="+disk_group_name/mydir/wallets"
   ```

4. As a user with the `SYSDBA` administrative privilege, run the `ALTER DISKGROUP` statements to create the necessary directories.

   You must perform this step because the database server cannot automate the creation of these directories, since the location chosen for `WALLET_ROOT` is not compliant with the Oracle Managed Files guideline (that is, it does not have the `OMF` component included in it in uppercase letters).

   a. Find the PDB GUID of the PDB that will store the keystore, as follows:

      ```
      SELECT GUID FROM DBA_PDBS WHERE PDB_NAME = 'pdb name';
      ```

   b. Include the PDB GUID in the following `ALTER DISKGROUP` statements to create the necessary directories for the isolated mode PDB within the `WALLET_ROOT` location. For example, assuming the GUID is `4756C705E52A8768E053F82DC40A5329`:

      ```
      ALTER DISKGROUP "disk_group_name" ADD DIRECTORY
          '+disk_group_name/mydir/wallets/4756C705E52A8768E053F82DC40A5329'

      ALTER DISKGROUP "disk_group_name" ADD DIRECTORY
          '+disk_group_name/mydir/wallets/4756C705E52A8768E053F82DC40A5329/tde';
      ```

7.1.14 Managing Updates to the PKCS#11 Library

Periodically, you may need to update the endpoint shared PKCS#11 library.

7.1.14.1 About Managing Updates to the PKCS#11 Library

The Oracle Database uses Oracle Key Vault's PKCS#11 endpoint shared library to retrieve the TDE master encryption key from Oracle Key Vault.

To switch an Oracle database over to an updated PKCS#11 shared library, you must run the following statement:

```
ADMINISTER KEY MANAGEMENT SWITCHOVER TO LIBRARY
'updated_fully_qualified_file_name_of_library' FOR ALL CONTAINERS;
```

Note the following:

- The path of the updated fully qualified file name must begin with `/opt/oracle/extapi/64/pkcs11/`. The path provided in the fully qualified file name will be validated to ensure that each directory is owned by `root` and is not writable by `group` or `other` (the permissions of each directory should look like `drwxr-xr-x root root`), and that none of the components of the path are symbolic links. If you receive an `ORA-02097: parameter cannot be modified because specified value is invalid` or `ORA-46702: failed to switch over the PKCS#11 library` error, then check the trace file for details regarding the path validation failure.

- Run this command from the root container database (`CDB$ROOT`). The PKCS#11 library is switched for the root container database and all PDBs that currently use the PKCS#11 library.
- Be aware that after you switch over to an updated PKCS#11 library, there may be a temporary decrease in TDE performance while the internal state of the PKCS#11 library is re-established, due to the internal state being lost during the library switchover.

7.1.14.2 Switching Over to an Updated PKCS#11 Library

When an updated PKCS#11 endpoint shared library is available, you can switch over to the updated library without incurring any database downtime.

1. Log in to the CDB root as a user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.
2. Perform the following steps.

   The following steps assume that the current Oracle Key Vault version is 21.5 and that you are upgrading to version 21.6. These versions are for illustrative purposes only.

   a. Create appropriate directories under `/opt/oracle/extapi/64/`. For example, for the Oracle Key Vault client software versions 21.5 and 21.6:

   ```
   $ sudo sh -c 'mkdir -pvm755 /opt/oracle/extapi/64/pkcs11/okv/lib/
   {21.5,21.6}'
   ```

   b. Run the Oracle Key Vault 21.5 client root script to copy the Oracle Key Vault PKCS#11 library into the legacy directory:

   ```
   $ sudo sh -c '/etc/ORACLE/KEYSTORES/finance/okv/bin/root.sh'
   ```

   c. Move the Oracle Key Vault PKCS#11 library into the correct version-depending directory:

   ```
   $ sudo sh -c 'mv -v
   /opt/oracle/extapi/64/hsm/oracle/1.0.0/liborapkcs.so
   /opt/oracle/extapi/64/pkcs11/okv/lib/21.5/'
   ```

   d. Confirm that the copy operation succeeed.

   ```
   $ tree -np /opt/oracle/extapi/64
   ```

   e. As a user with the `ALTER SYSTEM` privilege, set the static initialization parameter `PKCS11_LIBRARY_LOCATION`.

   ```
   ALTER SYSTEM SET PKCS11_LIBRARY_LOCATION = '/opt/oracle/extapi/64/
   pkcs11/okv/lib/21.5/liborapkcs.so' SCOPE = SPFILE SID = '*';
   ```

   After a database restart, your database is now ready to switch to new PKCS#11 libraries without DB downtime.

3. After you upgrade Oracle Key Vault to the new release (for example, 21.6), copy the new PKCS#11 library from the legacy directory into the following location:

   ```
   $ sudo sh -c 'mv -v /opt/oracle/extapi/64/hsm/oracle/1.0.0/
   liborapkcs.so /opt/oracle/extapi/64/pkcs11/okv/lib/21.6/'
   ```

4. Switch over to the updated PKCS#11 library.

```
ADMINISTER KEY MANAGEMENT SWITCHOVER TO LIBRARY
'updated_fully_qualified_file_name_of_library' FOR ALL CONTAINERS;
```

For example:

```
ADMINISTER KEY MANAGEMENT SWITCHOVER TO LIBRARY '/opt/oracle/extapi/64/
pkcs11/okv/lib/21.6/liborapkcs.so' FOR ALL CONTAINERS;
```

In this example, the library path uses the convention `/opt/oracle/extapi/64/pkcs11/vendor/lib/version/`. Oracle recommends that you have at minimum `version` in the library path because it will help in provisioning more libraries under `/opt/oracle/extapi/64/pkcs11`, which is needed for the library switchover operation.

If the database uses `SPFILE` to manage its parameters, then the library is switched. If the database uses `PFILE`, then the configuration is not changed, although the system does switch over to the updated PKCS#11 library. Otherwise any restart of the database instance would cause it to revert to using the earlier PKCS#11 library.

7.1.15 Backup and Recovery of Encrypted Data

For TDE wallets, you cannot access encrypted data without the TDE master encryption key.

Because the TDE master encryption key is stored in the TDE wallet, you should periodically back up the TDE wallet in a secure location. You must back up a copy of the TDE wallet whenever you set a new TDE master encryption key or perform any operation that writes to the TDE wallet.

Do not back up the TDE wallet in the same location as the encrypted data. Back up the TDE wallet separately. This is especially true when you use the auto-login TDE wallet, which does not require a password to open. In case the backup tape is lost, a malicious user should not be able to get both the encrypted data and the TDE wallet.

Oracle Recovery Manager (Oracle RMAN) does not back up the TDE wallet as part of the database backup. When using a media manager such as Oracle Secure Backup with Oracle RMAN, Oracle Secure Backup automatically excludes auto-open TDE wallets (the `cwallet.sso` files). However, it does not automatically exclude encryption TDE wallets (the `ewallet.p12` files). It is a good practice to add the following `exclude` data set statement to your Oracle Secure Backup configuration:

```
exclude name *.p12
```

This setting instructs Oracle Secure Backup to exclude the encryption TDE wallet from the backup set.

If you lose the TDE wallet that stores the TDE master encryption key, then you can restore access to encrypted data by copying the backed-up version of the TDE wallet to the appropriate location. If you archived the restored TDE wallet after the last time that you reset the TDE master encryption key, then you do not need to take any additional action.

If the restored TDE wallet does not contain the most recent TDE master encryption key, then you can recover old data up to the point when the TDE master encryption key was reset by rolling back the state of the database to that point in time. All of the modifications to encrypted columns after the TDE master encryption key was reset are lost.

Related Topics

- *Oracle Database Backup and Recovery User's Guide*

7.1.16 Dangers of Deleting TDE Wallets

Oracle strongly recommends that you do not delete TDE wallets.

If a TDE wallet becomes overly full, any TDE master encryption key other than the currently active TDE master encryption key can be moved to a new TDE wallet to reduce the overall size of the TDE wallet, but it is important to keep a backup of the old and new TDE wallets because even though the keys have been moved out of the currently active TDE wallet, they may still be needed by other Oracle features, such as Oracle Recovery Manager backup operations. (See Related Topics at the end of this topic for a listing of features that are affected by deleted TDE wallets.)

Deleting a TDE wallet that still contains keys is particularly dangerous if you have configured Transparent Data Encryption and the TDE wallet is in use. You can find if a TDE wallet is in use by querying the `STATUS` column of the `V$ENCRYPTION_WALLET` view after you open the TDE wallet. How you should proceed depends on whether you are using united mode or isolated mode.

- In isolated mode, if the `STATUS` column of the `V$ENCRYPTION_WALLET` is `OPEN_NO_MASTER_KEY`, then it is safe to archive and later delete the this TDE wallet, because there are no keys in it.
- In united mode, you must run the query of `V$ENCRYPTION_WALLET` from the `CDB$ROOT`, not the PDB. If you run the query in a PDB that does not yet have a key set, then the `STATUS` is `OPEN_NO_MASTER_KEY`. However, this can be misleading, because a key could have been set in the `CDB$ROOT`. After you run the query in the root and if the `STATUS` is `OPEN_NO_MASTER_KEY`, then you can safely archive and later delete the TDE wallet.

The reason that you should be cautious when moving keys out of the currently-active TDE keystore is that this wallet may contain keys that are still needed by the database (even though the TDE master encryption key has been rekeyed). Deleting the TDE wallet deletes these keys, and could result in the loss of encrypted data. Even if you decrypted all of the data in your database, you still should not delete the TDE wallet, because doing so could still hamper the normal functioning of the Oracle database. This is because a TDE master encryption key in the TDE wallet can also be required for other Oracle Database features. (See Related Topics at the end of this topic for a listing of features that are affected by deleted TDE wallets.)

Even after you performed TDE keystore migration (which rekeys in such a way that the location of your currently-active TDE master encryption key changes place between your TDE wallet and your external keystore), you still should not delete your original TDE wallet. The keys in the original TDE wallet may be needed at a later time (for example, when you must recover an offline encrypted tablespace). Even if all online tablespaces are not encrypted, the key may still be in use.

The exception is in the case of software auto-login (or local auto-login) TDE wallets. If you do not want to use this type of TDE wallet, then ideally you should move it to a secure directory. Only delete an auto-login TDE wallet if you are sure that it was created from a specific password-based TDE wallet because an auto-login TDE wallet is always based on an ordinary TDE wallet. The TDE wallet should be available and known.

If you must delete a TDE wallet, then do so with great caution. You must first move the keys within the TDE wallet to a new TDE wallet by using the `ADMINISTER KEY MANAGEMENT MOVE KEYS TO NEW KEYSTORE` statement.

Related Topics

- Features That Are Affected by Deleted Keystores
  Some features can be adversely affected if a keystore is deleted and a TDE master encryption key residing in that keystore is later needed.
- Moving a TDE Master Encryption Key into a New Keystore in United Mode
  In united mode, you can move an existing TDE master encryption key into a new keystore from an existing password-based TDE wallet.
- Moving a TDE Master Encryption Key into a New Keystore in Isolated Mode
  In isolated mode, you can move an existing TDE master encryption key into a new TDE wallet from an existing password TDE wallet.

7.1.17 Features That Are Affected by Deleted Keystores

Some features can be adversely affected if a keystore is deleted and a TDE master encryption key residing in that keystore is later needed.

Before you delete a keystore, consider the impact that the deletion will have in the event that you need the any TDE master encryption key in the TDE keystore at a later time. The following features and activities are affected:

- Offlined tablespace operations
- Oracle Secure Backup operations
- Media recovery and block media recovery operations
- Point-in-time recovery operations
- Physical and logical Oracle Data Guard standby operations
- Golden Gate operations
- Oracle Streams operations
- Oracle Recovery Manager operations, including restoring Oracle Recovery Manager backups
- Applying archived redo logs to a database during database crash recovery operations
- Database online block recovery. (Online block recovery implies that the database is still open. Deleting a wallet in an open database with encrypted tablespaces will cause additional problems other than those associated with online block recovery.) These problems can include the following:
  - Encrypted online data in encrypted tablespaces would no longer be decrypted. Encrypted metadata in the `SYSTEM`, `UNDO`, and `TEMP` tablespaces would no longer be decrypted. You will no longer have control over what metadata is encrypted or where that metadata can reside.
  - Buffered data or metadata needs to be encrypted before it can be written back to the disk, but if the wallet is deleted, then the buffered data or metadata would no longer be encrypted. This could cause redo generation to fail, and the DBWR background process would not be able to write the data, which would possibly lead to a database instance failure.
  - After a database instance failure, the database instance recovery and database crash recovery would fail, leading to the database not being able to be restarted.

Related Topics

- Dangers of Deleting TDE Wallets
  Oracle strongly recommends that you do not delete TDE wallets.

7.2 Managing the TDE Master Encryption Key

You can manage the TDE master encryption key in several ways.

7.2.1 TDE Master Encryption Key Attribute Management

TDE master encryption key attributes store information about the TDE master encryption key.

7.2.1.1 TDE Master Encryption Key Attributes

TDE master encryption key attributes include detailed information about the TDE master encryption key.

The information contains the following types:

- **Key time stamp information:** Internal security policies and compliance policies usually determine the key rekeying frequency. You should expire keys when they reach the end of their lifetimes and then generate new keys. Time stamp attributes such as key creation time and activation time help you to determine the key age accurately, and automate key generation.

  The `V$ENCRYPTION_KEYS` view includes columns such as `CREATION_TIME` and `ACTIVATION_TIME`. See *Oracle Database Reference* for a complete description of the `V$ENCRYPTION_KEYS` view.

- **Key owner information:** Key owner attributes help you to determine the user who created or activated the key. These attributes can be important for security, auditing, and tracking purposes. Key owner attributes also include key use information, such as whether the key is used for standalone TDE operations or used in a multitenant environment.

  The `V$ENCRYPTION_KEYS` view includes columns such as `CREATOR`, `CREATOR_ID`, `USER`, `USER_ID`, and `KEY_USE`.

- **Key source information:** Keys often must be moved between databases for operations such as import-export operations and Data Guard-related operations. Key source attributes enable you to track the origin of each key. You can track whether a key was created locally or imported, and the database name and instance number of the database that created the key. In a multitenant environment, you can track the PDB where the key was created.

  The `V$ENCRYPTION_KEYS` view includes columns such as `CREATOR_DBNAME`, `CREATOR_DBID`, `CREATOR_INSTANCE_NAME`, `CREATOR_INSTANCE_NUMBER`, `CREATOR_PDBNAME`, and so on.

- **Key usage information:** Key usage information determines the database or PDB where the key is being used. It also helps determine whether a key is in active use or not.

  The `V$ENCRYPTION_KEYS` view includes columns such as `ACTIVATING_DBNAME`, `ACTIVATING_DBID`, `ACTIVATING_INSTANCE_NAME`, `ACTIVATING_PDBNAME`, and so on.

- **User-defined information and other information:** When creating a key, you can tag it with information using the `TAG` option. Each key contains important information such as whether or not it has been backed up.

  The `V$ENCRYPTION_KEYS` view includes columns such as `KEY_ID`, `TAG`, and other miscellaneous columns, for example `BACKED_UP`.

> **Note:**
>
> TDE Master Key Attributes and Tag are only supported with Oracle Key Vault and Oracle Cloud Infrastructure (OCI) Key Management Service (KMS).

7.2.1.2 Finding the TDE Master Encryption Key That Is in Use

A TDE master encryption key that is in use is the encryption key that was activated most recently for the database.

- To find the TDE master encryption key, query the `V$ENCRYPTION_KEYS` dynamic view.

  For example:

```
SELECT KEY_ID
FROM V$ENCRYPTION_KEYS
WHERE ACTIVATION_TIME = (SELECT MAX(ACTIVATION_TIME)
                         FROM V$ENCRYPTION_KEYS
                         WHERE ACTIVATING_DBID = (SELECT DBID FROM V$DATABASE));
```

7.2.2 Creating Custom TDE Master Encryption Key Attributes for Reports

Custom TDE master encryption key attributes enable you to defined attributes that are specific to your needs.

7.2.2.1 About Creating Custom Attribute Tags

Attribute tags enable you to monitor specific activities users perform, such as accessing a particular terminal ID.

By default, Oracle Database defines a set of attributes that describe various characteristics of the TDE master encryption keys that you create, such as the creation time, database in which the TDE master encryption key is used, and so on. These attributes are captured by the `V$ENCRYPTION_KEY` dynamic view.

You can create custom attributes that can be captured by the `TAG` column of the `V$ENCRYPTION_KEYS` dynamic view. This enables you to define behaviors that you may want to monitor, such as users who perform activities on encryption keys. The tag can encompass multiple attributes, such as session IDs from a specific terminal.

After you create the tag for a TDE master encryption key, its name should appear in the `TAG` column of the `V$ENCRYPTION_KEYS` view for that TDE master encryption key. If you create a tag for the secret, then the tag appears in the `SECRET_TAG` column of the `V$CLIENT_SECRETS` view. If you create a secret with a tag, then the tag appears in the `SECRET_TAG` column of the `V$CLIENT_SECRETS` view.

7.2.2.2 Creating a Custom Attribute Tag

To create a custom attribute tag, you must use the `SET TAG` clause of the `ADMINISTER KEY MANAGEMENT` statement.

1. Connect to the united mode CDB root or isolated mode PDB as a user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.
2. If necessary, query the `TAG` column of the `V$ENCRYPTION_KEY` dynamic view to find a listing of existing tags for the TDE master encryption keys.

When you create a new tag for a TDE master encryption key, it overwrites the existing tag for that TDE master encryption key.

3. Create the custom attribute tag by using the following syntax:

```
ADMINISTER KEY MANAGEMENT SET TAG 'tag'
FOR 'master_key_identifier'
[FORCE KEYSTORE]
IDENTIFIED BY [EXTERNAL STORE | keystore_password]
[WITH BACKUP [USING 'backup_identifier']];
```

In this specification

- *tag* is the associated attributes or information that you define. Enclose this information in single quotation marks (' ').
- *master_key_identifier* identifies the TDE master encryption key for which the *tag* is set. To find a list of TDE master encryption key identifiers, query the KEY_ID column of the V$ENCRYPTION_KEYS dynamic view.
- FORCE KEYSTORE temporarily opens the password-protected TDE wallet for this operation. You must open the TDE wallet for this operation.
- IDENTIFIED BY can be one of the following settings:
  - EXTERNAL STORE uses the keystore password stored in the external store to perform the keystore operation.
  - *keystore_password* is the password that was used to create the keystore.
- *backup_identifier* defines the tag values. Enclose this setting in single quotation marks (' ') and separate each value with a colon.

For example, to create a tag that uses two values, one to capture a specific session ID and the second to capture a specific terminal ID:

```
ADMINISTER KEY MANAGEMENT SET ENCRYPTION KEY
USING TAG 'sessionid=3205062574:terminal=xcvt'
IDENTIFIED BY keystore_password
WITH BACKUP;

keystore altered.
```

Both the session ID (3205062574) and terminal ID (xcvt) can derive their values by using either the SYS_CONTEXT function with the USERENV namespace, or by using the USERENV function.

7.2.3 Setting or Rekeying the TDE Master Encryption Key in the Keystore

You can set or rekey the TDE master encryption key for both TDE wallets and external keystores.

7.2.3.1 About Setting or Rekeying the TDE Master Encryption Key in the Keystore

You can set or rekey the TDE master encryption key for both software password-based and external keystores.

The TDE master encryption key is stored in an external security module (keystore), and it is used to protect the TDW table keys and tablespace encryption keys. By default, the TDE master encryption key is a system-generated random value created by Transparent Data Encryption (TDE).

Use the `ADMINISTER KEY MANAGEMENT` statement to set or reset (`REKEY`) the TDE master encryption key. When the master encryption key is set, then TDE is considered enabled and cannot be disabled.

Before you can encrypt or decrypt database columns or tablespaces, you must generate a TDE master encryption key. Oracle Database uses the same TDE master encryption key for both TDE column encryption and TDE tablespace encryption. The instructions for setting a software or hardware TDE master encryption key explain how to generate a tDE master encryption key.

Related Topics

- Step 3: Set the TDE Master Encryption Key in the TDE Wallet
  Once the TDE wallet is open, you can set a TDE master encryption key for it.
- Step 3: Set the TDE Master Encryption Key in Oracle Key Vault
  After you have opened the connection to Oracle Key Vault, you are ready to set the TDE master encryption key.

7.2.3.2 Creating, Tagging, and Backing Up a TDE Master Encryption Key

The `ADMINISTER KEY MANAGEMENT` statement enables you to create, tag, and back up a TDE master encryption key.

- Create and back up the TDE master encryption key, and apply a tag, by using the following syntax:

```
ADMINISTER KEY MANAGEMENT SET ENCRYPTION KEY
[USING TAG 'tag']
[FORCE KEYSTORE]
IDENTIFIED BY [EXTERNAL STORE | keystore_password]
WITH BACKUP [USING 'backup_identifier'];
```

  In this specification:

  - *tag* is the tag that you want to create. Enclose this tag in single quotation marks (`' '`).
  - `FORCE KEYSTORE` temporarily opens the password-protected TDE wallet for this operation. You must open the TDE wallet for this operation.
  - `IDENTIFIED BY` can be one of the following settings:
    - `EXTERNAL STORE` uses the keystore password stored in the external store to perform the keystore operation.
    - *keystore_password* is either *TDE_wallet_password* or *external_key_manager_password*. You must enclose the password string in double quotation marks (`" "`).
  - `WITH BACKUP` backs the TDE master encryption key up in the same location as the key, as identified by the `WRL_PARAMETER` column of the `V$ENCRYPTION_WALLET` view. To find the `WRL_PARAMETER` values for all of the database instances, query the `GV$ENCRYPTION_WALLET` view.

    You must back up password-based TDE wallets. You do not need to use it for auto-login or local auto-login TDE wallets. Optionally, include the `USING` *backup_identifier* clause to add a description of the backup. Enclose this identifier in single quotation marks (`' '`).

  For example:

```
ADMINISTER KEY MANAGEMENT SET ENCRYPTION KEY
USING TAG 'backups"
IDENTIFIED BY keystore_password
WITH BACKUP USING 'hr.emp_key_backup';

keystore altered.
```

Oracle Database uses the keystore in the keystore location specified by the `WALLET_ROOT` parameter in the initialization parameter file to store the TDE master encryption key.

Related Topics

- Creating Custom TDE Master Encryption Key Attributes for Reports
  Custom TDE master encryption key attributes enable you to defined attributes that are specific to your needs.

7.2.3.3 About Rekeying the TDE Master Encryption Key

Oracle Database uses a unified TDE Master Encryption Key for both TDE column encryption and TDE tablespace encryption.

When you rekey the TDE master encryption key for TDE column encryption, the TDE Master Encryption Key for TDE tablespace encryption also is rekeyed. Rekey the TDE Master Encryption Key only if it was compromised or as per the security policies of the organization. This process deactivates the previous TDE master encryption key.

For better security and to meet compliance regulations, periodically rekey the TDE master encryption key. This process deactivates the previous TDE master encryption key, creates a new TDE master encryption key, and then activates it. You can check the keys that were created recently by querying the `CREATION_TIME` column in the `V$ENCRYPTION_KEYS` view. To find the keys that were activated recently, query the `ACTIVATION_TIME` column in the `V$ENCRYPTION_KEYS` view.

You cannot change the TDE master encryption key or rekey a TDE master encryption key for an auto-login keystore. Because auto-login keystores do not have a password, an administrator or a privileged user can change the keys without the knowledge of the security officer. However, if both the auto-login and the password-based keystores are present in the configured location (as set in the `sqlnet.ora` file), then when you rekey the TDE master encryption key, a TDE master encryption key is added to both the auto-login and password-based keystores. If the auto-login keystore is in use in a location that is different from that of the password-based keystore, then you must re-create the auto-login keystore.

Do not perform a rekey operation of the master key concurrently with an online tablespace rekey operation. You can find if an online tablespace is in the process of being TDE Master Encryption Keyed by issuing the following query:

```
SELECT TS#,ENCRYPTIONALG,STATUS FROM V$ENCRYPTED_TABLESPACES;
```

A status of `REKEYING` means that the corresponding tablespace is still being rekeyed.

> **Note:**
>
> You cannot add new information to auto-login keystores separately.

7.2.3.4 Rekeying the TDE Master Encryption Key

You can use the `ADMINISTER KEY MANAGEMENT` statement to rekey a TDE master encryption key.

1. Connect to the united mode CDB root or isolated mode PDB as a user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.
2. If you are rekeying the TDE master encryption key for a keystore that has auto login enabled, then ensure that both the auto login keystore, identified by the `.sso` file, and the encryption keystore, identified by the `.p12` file, are present.

   You can find the location of these files by querying the `WRL_PARAMETER` column of the `V$ENCRYPTION_WALLET` view. To find the `WRL_PARAMETER` values for all of the database instances, query the `GV$ENCRYPTION_WALLET` view.
3. Rekey the TDE master encryption key by using the following syntax:

   ```
   ADMINISTER KEY MANAGEMENT SET [ENCRYPTION] KEY
   [FORCE KEYSTORE]
   [USING TAG 'tag_name']
   IDENTIFIED BY [EXTERNAL STORE | keystore_password]
   [WITH BACKUP [USING 'backup_identifier']];
   ```

   In this specification:

   - *tag* is the associated attributes and information that you define. Enclose this setting in single quotation marks (`' '`).
   - `FORCE KEYSTORE` temporarily opens the password-protected TDE wallet for this operation. You must open the TDE wallet for this operation.
   - `IDENTIFIED BY` can be one of the following settings:
     - `EXTERNAL STORE` uses the keystore password stored in the external store to perform the keystore operation.
     - *keystore_password* is the mandatory keystore password that you created when you created the keystore in Step 1: Create the TDE Wallet.
   - `WITH BACKUP` creates a backup of the keystore. You must use this option for password-based and external keystores. Optionally, you can use the `USING` clause to add a brief description of the backup. Enclose this description in single quotation marks (`' '`). This identifier is appended to the named keystore file (for example, `ewallet_`*time-stamp*`_emp_key_backup.p12`). Follow the file naming conventions that your operating system uses.

   For example:

   ```
   ADMINISTER KEY MANAGEMENT SET KEY
   FORCE KEYSTORE
   IDENTIFIED BY keystore_password
   WITH BACKUP USING 'emp_key_backup';

   keystore altered.
   ```

Related Topics

- Step 2: Open the TDE Wallet
  Depending on the type of TDE wallet you create, you must manually open the wallet before you can use it.

- Step 2: Open the Connection to Oracle Key Vault
  After you have configured the database to use Oracle Key Vault for TDE key management, you must open the connection to Oracle Key Vault before you can use it.

7.2.3.5 Changing the TDE Master Encryption Key for a Tablespace

You can use the `ENCRYPT` and `REKEY` clauses of the `ALTER TABLESPACE` statement to encrypt a tablespace.

1. Connect to the united mode CDB root or isolated mode PDB as a user who has been granted administrative privileges.
2. Ensure that the tablespace open in read-write mode.

   You can query the `STATUS` column of the `V$INSTANCE` dynamic view to find if a database is open and the `OPEN_MODE` column of the `V$DATABASE` view to find if it in read-write mode.
3. If necessary, open the database in read-write mode.

   ```
   ALTER DATABASE OPEN READ WRITE;
   ```
4. Run the `ALTER TABLESPACE` SQL statement to encrypt the tablespace.

   If the tablespace has not yet been encrypted, then use the `ENCRYPT` clause:

   ```
   ALTER TABLESPACE encrypt_ts ENCRYPTION USING 'AES256' ENCRYPT;
   ```

   To change the encryption of the `SYSTEM`, `SYSAUX`, or `UNDO` tablespace, you must rekey the tablespace online. Use the `ONLINE` and `REKEY` clauses. For example, to change the encryption algorithm of the `SYSTEM` tablespace:

   ```
   ALTER TABLESPACE SYSTEM ENCRYPTION ONLINE USING 'AES256' REKEY;
   ```

7.2.4 Exporting and Importing the TDE Master Encryption Key

You can export and import the TDE master encryption key in different ways.

7.2.4.1 About Exporting and Importing the TDE Master Encryption Key

Oracle Database features such as transportable tablespaces and Oracle Data Pump move data that is possibly encrypted between databases.

These are some common scenarios in which you can choose to export and import TDE master encryption keys to move them between source and target keystores. For Data Guard (Logical Standby), you must copy the keystore that is in the primary database to the standby database. Instead of merging the primary database keystore with the standby database, you can export the TDE master encryption key that is in use and then import it to the standby database. Moving transportable tablespaces that are encrypted between databases requires that you export the TDE master encryption key at the source database and then import it into the target database.

7.2.4.2 About Exporting TDE Master Encryption Keys

You can use `ADMINISTER KEY MANAGEMENT EXPORT` to export TDE master encryption keys from a keystore, and then import them into another keystore.

A TDE master encryption key is exported together with its key identifier and key attributes. The exported keys are protected with a password (secret) in the export file.

You can specify the TDE master encryption keys to be exported by using the `WITH IDENTIFIER` clause of the `ADMINSITER KEY MANAGENT EXPORT` statement. To export the TDE master encryption keys, you can either specify their key identifiers as a comma-separated list, or you can specify a query that enumerates their key identifiers. Be aware that Oracle Database runs the query determining the key identifiers within the current user's rights and not with definer's rights.

If you omit the `WITH IDENTIFER` clause, then all of the TDE master encryption keys of the database are exported.

Related Topics

- Exporting and Importing Master Encryption Keys for a PDB in Isolated Mode
  In isolated mode, the `EXPORT` and `IMPORT` clauses of `ADMINISTER KEY MANAGEMENT EXPORT` can export or import master encryption keys for a PDB.

7.2.4.3 Exporting a TDE Master Encryption Key

The `ADMINISTER KEY MANAGEMENT` statement with the `EXPORT [ENCRYPTION] KEYS WITH SECRET` clause exports a TDE master encryption key.

- Export the TDE master encryption keys by using the following syntax:

```
ADMINISTER KEY MANAGEMENT EXPORT [ENCRYPTION] KEYS
WITH SECRET "export_secret"
TO 'file_path'
[FORCE KEYSTORE]
IDENTIFIED BY [EXTERNAL STORE | keystore_password]
[WITH IDENTIFIER IN 'key_id1', 'key_id2', 'key_idn' | (SQL_query)];
```

  In this specification:

  - *export_secret* is a password that you can specify to encrypt the export the file that contains the exported keys. Enclose this secret in double quotation marks (" "), or you can omit the quotation marks if the secret has no spaces.
  - *file_path* is the complete path and name of the file to which the keys must be exported. Enclose this path in single quotation marks (`' '`). You can export to regular file systems only.
  - `FORCE KEYSTORE` temporarily opens the password-protected TDE wallet for this operation. You must open the TDE wallet for this operation.
  - `IDENTIFIED BY` can be one of the following settings:
    - `EXTERNAL STORE` uses the keystore password stored in the external store to perform the keystore operation.
    - *TDE_wallet_password* is the password of the keystore containing the keys.
  - *key_id1*, *key_id2*, *key_idn* is a string of one or more TDE master encryption key identifiers for the TDE master encryption key being exported. Separate each key identifier with a comma and enclose each of these key identifiers in single quotation marks (`' '`). To find a list of TDE master encryption key identifiers, query the `KEY_ID` column of the `V$ENCRYPTION_KEYS` dynamic view.
  - *SQL_query* is a query that fetches a list of the TDE master encryption key identifiers. It should return only one column which contains the TDE master encryption key identifiers. This query is run with current user rights.

7.2.4.4 Example: Exporting a TDE Master Encryption Key by Using a Subquery

The `ADMINISTER KEY MANAGEMENT EXPORT ENCRYPTION KEYS` statement can export a TDE master encryption key by using a subquery.

Example 7-2 shows how to export TDE master encryption keys whose identifiers are fetched by a query to a file called `export.exp`. The TDE master encryption keys in the file are encrypted using the secret `my_secret`. The `SELECT` statement finds the identifiers for the TDE master encryption keys to be exported.

Example 7-1 Exporting a List of TDE Master Encryption Key Identifiers to a File

```
ADMINISTER KEY MANAGEMENT EXPORT ENCRYPTION KEYS
WITH SECRET "my_secret"
TO '/TDE/export.exp'
FORCE KEYSTORE
IDENTIFIED BY password
WITH IDENTIFIER IN 'AdoxnJ0uH08cv7xkz83ovwsAAAAAAAAAAAAAAAAAAAAAAAAAAAA',
'AW5z3CoyKE/yv3cNT5CWCXUAAAAAAAAAAAAAAAAAAAAAAAAAAAAA';

keystore altered.
```

7.2.4.5 Example: Exporting a List of TDE Master Encryption Key Identifiers to a File

The `ADMINISTER KEY MANAGEMENT EXPORT ENCRYPTION KEYS WITH SECRET` statement can export a list of TDE master encryption key identifiers to a file.

Example 7-1 shows how to export TDE master encryption keys by specifying their identifiers as a list, to a file called `export.exp`. TDE master encryption keys in the file are encrypted using the secret `my_secret`. The identifiers of the TDE master encryption key to be exported are provided as a comma-separated list.

Example 7-2 Exporting TDE Master Encryption Key Identifiers by Using a Subquery

```
ADMINISTER KEY MANAGEMENT EXPORT ENCRYPTION KEYS
WITH SECRET "my_secret" TO '/etc/TDE/export.exp'
FORCE KEYSTORE
IDENTIFIED BY password
WITH IDENTIFIER IN (SELECT KEY_ID FROM V$ENCRYPTION_KEYS WHERE ROWNUM <3);

keystore altered.
```

7.2.4.6 Example: Exporting All TDE Master Encryption Keys of the Database

The `ADMINISTER KEY MANAGEMENT EXPORT ENCRYPTION KEYS` SQL statement can export all TDE master encryption keys of a database.

Example 7-3 shows how to export all of the TDE master encryption keys of the database to a file called `export.exp`. The TDE master encryption keys in the file are encrypted using the secret `my_secret`.

Example 7-3 Exporting All of the TDE Master Encryption Keys of the Database

```
ADMINISTER KEY MANAGEMENT EXPORT ENCRYPTION KEYS
WITH SECRET "my_secret" TO '/etc/TDE/export.exp'
FORCE KEYSTORE
IDENTIFIED BY password;

keystore altered.
```

7.2.4.7 About Importing TDE Master Encryption Keys

The `ADMINISTER KEY MANAGEMENT IMPORT` statement can import exported TDE master encryption keys from a key export file into a target keystore.

You cannot re-import TDE master encryption keys that have already been imported.

Related Topics

- Exporting and Importing Master Encryption Keys for a PDB in Isolated Mode
  In isolated mode, the `EXPORT` and `IMPORT` clauses of `ADMINISTER KEY MANAGEMENT EXPORT` can export or import master encryption keys for a PDB.

7.2.4.8 Importing a TDE Master Encryption Key

The `ADMINISTER KEY MANAGEMENT` statement with the `IMPORT [ENCRYPTION] KEYS WITH SECRET` clause can import a TDE master encryption key.

- Use the following syntax to import a TDE master encryption key:

```
ADMINISTER KEY MANAGEMENT IMPORT [ENCRYPTION] KEYS
WITH SECRET "import_secret"
FROM 'file_name'
[FORCE KEYSTORE]
IDENTIFIED BY [EXTERNAL STORE | keystore_password]
[WITH BACKUP [USING 'backup_identifier']];
```

  In this specification:

  - *import_secret* is the same password that was used to encrypt the keys during the export operation. Enclose this secret in double quotation marks (" "), or you can omit the quotation marks if the secret has no spaces.
  - *file_name* is the complete path and name of the file from which the keys need to be imported. Enclose this setting in single quotation marks (' ').
  - `FORCE KEYSTORE` temporarily opens the password-protected TDE wallet for this operation. You must open the TDE wallet for this operation.
  - `IDENTIFIED BY` can be one of the following settings:
    - `EXTERNAL STORE` uses the keystore password stored in the external store to perform the keystore operation.
    - *TDE_wallet_password* is the password of the TDE wallet where the keys are being imported.
  - `WITH BACKUP` must be used in case the target keystore was not backed up before the import operation. *backup_identifier* is an optional string that you can provide to identify the keystore backup. Enclose this setting in single quotation marks (' ').

7.2.4.9 Example: Importing a TDE Master Encryption Key

You can use the `ADMINISTER KEY MANAGEMENT IMPORT KEYS` SQL statement to import a TDE master encryption key.

Example 7-4 shows how to import the TDE master encryption key identifiers that are stored in the file `export.exp` and encrypted with the secret `my_secret`.

Example 7-4 Importing TDE Master Encryption Key Identifiers from an Export File

```
ADMINISTER KEY MANAGEMENT IMPORT KEYS
WITH SECRET "my_secret"
FROM '/etc/TDE/export.exp'
FORCE KEYSTORE
IDENTIFIED BY password WITH BACKUP;

keystore altered.
```

7.2.4.10 How Keystore Merge Differs from TDE Master Encryption Key Export or Import

The keystore merge operation differs from the TDE master encryption key export and import operations.

Even though both the `ADMINISTER KEY MANAGEMENT MERGE` statement and the `ADMINISTER KEY MANAGEMENT EXPORT` and `IMPORT` statements eventually move the TDE master encryption keys from one keystore to the next, there are differences in how these two statements function.

- The `MERGE` statement merges two keystores whereas the `EXPORT` and `IMPORT` statements export the keys to a file or import the keys from a file. The keystore is different from the export file, and the two cannot be used interchangeably. The export file is not a keystore and cannot be configured to be used with a database as a keystore. Similarly, the `IMPORT` statement cannot extract the TDE master encryption keys from the keystore.
- The `MERGE` statement merges all of the TDE master encryption keys of the specified keystores where as the `EXPORT` and `IMPORT` statements can be selective.
- The `EXPORT` and `IMPORT` statements require the user to provide both a location (`filepath`) and the file name of the export file, whereas the `MERGE` statement only takes in the location of the keystores.
- The file name of the keystores is fixed and is determined by the `MERGE` operation and can be either `ewallet.p12` or `cwallet.sso`. The file names for the export files used in the `EXPORT` the `IMPORT` statements are specified by the user.
- The keystores on Automatic Storage Management (ASM) disk groups or regular file systems can be merged with `MERGE` statements. The export files used in the `EXPORT` and the `IMPORT` statements can only be a regular operating system file and cannot be located on an ASM disk group.
- The keystores merged using the `MERGE` statement do not need to be configured or in use with the database. The `EXPORT` statement can only export the keys from a keystore that is configured and in use with the database and is also open when the export is done. The `IMPORT` statement can only import the keys into a keystore that is open, configured, and in use with the database.
- The `MERGE` statement never modifies the metadata associated with the TDE master encryption keys. The `EXPORT` and `IMPORT` operations can modify the metadata of the TDE master encryption keys when required, such as during a PDB plug operation.

7.2.5 Converting from ENCRYPTION_WALLET_LOCATION to WALLET_ROOT and TDE_CONFIGURATION

You can convert the wallet location from using the `sqlnet.ora` `ENCRYPTION_WALLET_LOCATION` parameter to using the `WALLET_ROOT` and `TDE_CONFIGURATION` parameters instead.

The conversion applies to either file systems or Oracle Automatic Storage Management (Oracle ASM) configurations. This procedure assumes that you have already set the `WALLET_ROOT` static initialization parameter and created a *`WALLET_ROOT`*`/tde` directory.

1. Copy all the wallets to the *`WALLET_ROOT`*`/tde` directory.
2. Restart the database.
3. Back up the database.
4. Connect to the CDB root as a user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.
5. Set the `TDE_CONFIGURATION` dynamic initialization parameter to `FILE`.

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

6. Rename the wallets from where you copied them.
7. Remove the old `ENCRYPTION_WALLET_LOCATION` configuration setting from the `sqlnet.ora` file.

7.2.6 Management of TDE Master Encryption Keys Using Oracle Key Vault

You can use Oracle Key Vault to manage and share TDE master encryption keys across an enterprise.

Oracle Key Vault securely stores the keys in a central repository, along with other security objects such as credential files and Java keystores, and enables you to share these objects with other TDE-enabled databases.

Related Topics

- Migration of Keystores to and from Oracle Key Vault
  You can use Oracle Key Vault to migrate both TDE wallets and external keystores to and from Oracle Key Vault.
- *Oracle Key Vault Administrator's Guide*

7.3 Transparent Data Encryption Data Dynamic and Data Dictionary Views

You can query a set of dynamic and data dictionary views to find more information about Transparent Data Encryption (TDE) data.

Table 7-2 describes these dynamic and data dictionary views.

Table 7-2 Transparent Data Encryption Related Views

View	Description
ALL_ENCRYPTED_COLUMNS	Displays encryption information about encrypted columns in the tables accessible to the current user
DICTIONARY_CREDENTIALS_ENCRYPT	Indicates if credential data in the SYS.LINK$ and SYS.SCHEDULER$_CREDENTIAL system tables is encrypted
DBA_ENCRYPTED_COLUMNS	Displays encryption information for all of the encrypted columns in the database
USER_ENCRYPTED_COLUMNS	Displays encryption information for encrypted table columns in the current user's schema
DBA_TABLESPACE_USAGE_METRICS	Describes tablespace usage metrics for all types of tablespaces, including permanent, temporary, and undo tablespaces
V$CLIENT_SECRETS	Lists the properties of the strings (secrets) that were stored in the keystore for various features (clients). In a multitenant environment, when you query this view in a PDB, then it displays information about keys that were created or activated for the current PDB. If you query this view in the root, then it displays this information about keys for all of the PDBs.
V$DATABASE_KEY_INFO	Displays information about the default encryption key that is used for the current database. The default is AES256.
V$ENCRYPTED_TABLESPACES	Displays information about the tablespaces that are encrypted
V$ENCRYPTION_KEYS	When used with keys that have been rekeyed with the ADMINISTER KEY MANAGEMENT statement, displays information about the TDE master encryption keys. In a multitenant environment, when you query this view in a PDB, it displays information about keys that were created or activated for the current PDB. If you query this view in the root, it displays this information about keys for all of the PDBs.
V$ENCRYPTION_WALLET	Displays information on the status of the keystore and the keystore location for TDE
V$RMAN_ENCRYPTION_ALGORITHMS	Displays supported encryption algorithms in the current PDB and is used by Oracle Recovery Manager (Oracle RMAN) to validate user-requested algorithms

Related Topics

- *Oracle Database Reference*

8
Administering United Mode

Administering united mode means managing the keystores, master encryption keys, and general Transparent Database Encryption (TDE) functionality.

8.1 Administering Keystores and Master Encryption Keys in United Mode

After you configure a keystore and master encryption key for use in united mode, you can perform tasks such as rekeying TDE master encryption keys.

8.1.1 Changing the Keystore Password in United Mode

You can change the password of either a TDE wallet or an external keystore only in the CDB root.

8.1.1.1 Changing the Password-Protected TDE Wallet Password in United Mode

To change the password of a password-protected TDE wallet in united mode, you must use the `ADMINISTER KEY MANAGEMENT` statement in the CDB root.

You can change this password at any time, as per the security policies, compliance guidelines, and other security requirements of your site. As part of the command to change the password, you will be forced to specify the `WITH BACKUP` clause, and thus forced to make a backup of the current TDE wallet. During the password change operation, Transparent Data Encryption operations such as encryption and decryption will continue to work normally. You can change this password at any time. You should change this password if you think it was compromised.

1. Connect to the CDB root as a common user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.
2. Use the following syntax to change the password for the keystore:

```
ADMINISTER KEY MANAGEMENT ALTER KEYSTORE PASSWORD
[FORCE KEYSTORE]
IDENTIFIED BY
old_TDE_wallet_password SET new_TDE_wallet_password
WITH BACKUP [USING 'backup_identifier'];
```

In this specification:

- `FORCE KEYSTORE` temporarily opens the password-protected TDE wallet for this operation if the TDE wallet is closed, if an auto-login TDE wallet is configured and is currently open, or if a password-protected TDE wallet is configured and is currently closed.
- *`old_TDE_wallet_password`* is the current password that you want to change.
- *`new_TDE_wallet_password`* is the new password.

- You do not need to include the `CONTAINER` clause because the password can only be changed locally, in the CDB root.

The following example creates a backup of the keystore and then changes the password:

```
ADMINISTER KEY MANAGEMENT ALTER KEYSTORE PASSWORD
IDENTIFIED BY old_password SET new_password
WITH BACKUP USING 'before_password_change';

keystore altered.
```

This example performs the same operation but uses the `FORCE KEYSTORE` clause in case the auto-login TDE wallet is in use or the password-protected TDE wallet is closed.

```
ADMINISTER KEY MANAGEMENT ALTER KEYSTORE PASSWORD
FORCE KEYSTORE
IDENTIFIED BY old_password SET new_password
WITH BACKUP USING 'before_password_change';

keystore altered.
```

Related Topics

- Performing Operations That Require a Keystore Password
  Many `ADMINISTER KEY MANAGEMENT` operations require access to a keystore password, for both TDE wallets and external keystores.

8.1.1.2 Changing the Password of an External Keystore in United Mode

To change the password of an external keystore, you must close the external keystore and then change the password from the external keystore management interface.

1. Connect to the CDB root as a common user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.
2. Close the external keystore.
   - Close the connection to the external key manager:

     ```
     ADMINISTER KEY MANAGEMENT SET KEYSTORE CLOSE
     IDENTIFIED BY "external_key_manager_password"|EXTERNAL STORE
     CONTAINER = ALL;
     ```

   - If the keystore was auto-opened by the database, then close the connection to the external key manager as follows:

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

3. Change the Oracle Key Vault password.

   ```
   WALLET_ROOT/okv/bin/okvutil changepwd -t wallet -l WALLET_ROOT/okv/ssl
   ```

4. Update the Oracle Key Vault password that enables an auto-open Oracle Key Vault keystore to use the new password that you set in step 3.

   ```
   ADMINISTER KEY MANAGEMENT UPDATE SECRET 'new_Oracle_Key_Vault_password'
   FOR CLIENT 'OKV-PASSWORD' TO [LOCAL] AUTO-LOGIN KEYSTORE 'WALLET_ROOT/tde';
   ```

5. Update the Oracle Key Vault password that enables `IDENTIFIED BY EXTERNAL STORE (IBES)` to the new password that you set in step 3.

```
ADMINISTER KEY MANAGEMENT UPDATE SECRET 'new_Oracle_Key_Vault_password'
FOR CLIENT 'OKV-PASSWORD' TO [LOCAL] AUTO-LOGIN KEYSTORE 'WALLET_ROOT/
tde_seps';
```

6. Open the connection to Oracle Key Vault.

```
ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN
IDENTIFIED BY EXTERNAL STORE;
```

Related Topics

- Performing Operations That Require a Keystore Password
  Many `ADMINISTER KEY MANAGEMENT` operations require access to a keystore password, for both TDE wallets and external keystores.

8.1.2 Backing Up a Password-Protected TDE Wallet in United Mode

The `BACKUP KEYSTORE` clause of the `ADMINISTER KEY MANAGEMENT` statement backs up a password-protected TDE wallet.

1. Connect to the CDB root as a common user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.
2. Back up the keystore by using the following syntax:

```
ADMINISTER KEY MANAGEMENT BACKUP KEYSTORE
[USING 'backup_identifier']
[FORCE KEYSTORE]
IDENTIFIED BY [EXTERNAL STORE | TDE_wallet_password]
[TO 'keystore_location'];
```

   In this specification:

   - `USING` *`backup_identifier`* is an optional string that you can provide to identify the backup. Enclose this identifier in single quotation marks (`' '`). This identifier is appended to the named keystore file (for example, `ewallet_`*`time-stamp`*`_emp_key_backup.p12`).
   - `FORCE KEYSTORE` temporarily opens the password-protected TDE wallet for this operation. You must open the TDE wallet for this operation.
   - `IDENTIFIED BY` is required for the `BACKUP KEYSTORE` operation on a password-protected keystore because although the backup is simply a copy of the existing keystore, the status of the TDE master encryption key in the password-protected keystore must be set to `BACKED UP` and for this change the keystore password is required.
   - *`keystore_location`* is the path at which the backup keystore is stored. This setting is restricted to the PDB when the PDB lockdown profile `EXTERNAL_FILE_ACCESS` setting is blocked in the PDB or when the `PATH_PREFIX` variable was not set when the PDB was created. If you do not specify the *`keystore_location`*, then the backup is created in the same directory as the original keystore. Enclose this location in single quotation marks (`' '`).
   - You do not need to include the `CONTAINER` clause because the keystore can only be backup up locally, in the CDB root.

The following example backs up a TDE wallet in the same location as the source keystore.

```
ADMINISTER KEY MANAGEMENT BACKUP KEYSTORE
USING 'hr.emp_keystore'
FORCE KEYSTORE
IDENTIFIED BY
TDE_wallet_password ;

keystore altered.
```

In the following version, the password for the keystore is external, so the `EXTERNAL STORE` clause is used.

```
ADMINISTER KEY MANAGEMENT BACKUP KEYSTORE
USING 'hr.emp_keystore'
FORCE KEYSTORE
IDENTIFIED BY EXTERNAL STORE;
```

After you run this statement, an `ewallet_`*identifier*`.p12` file (for example, `ewallet_`*time-stamp*`_hr.emp_keystore.p12`) appears in the TDE wallet backup location.

Related Topics

- Backing Up Password-Protected TDE Wallets
  When you back up a password-protected TDE wallet, you can create a backup identifier string to describe the backup type.

8.1.3 Closing Keystores in United Mode

You can close both TDE wallet and external keystores in united mode, unless the system tablespace is encrypted.

8.1.3.1 About Closing Keystores

After you open a keystore, it remains open until you shut down the database instance.

When you restart the database instance, then auto-login and local auto-login TDE wallets automatically open when required (that is, when the TDE master encryption key must be accessed). However, TDE wallet password-based and external keystores do not automatically open. You must manually open them again before you can use them.

When you close a TDE wallet or an external keystore, you disable all of the encryption and decryption operations on the database. Hence, a database user or application cannot perform any operation involving encrypted data until the TDE wallet or keystore is reopened.

When you re-open a TDE wallet keystore after closing it, its contents are reloaded back into the database. Thus, if the contents had been modified (such as during a migration), the database will have the latest TDE wallet or keystore contents.

When you run the `ALTER PLUGGABLE DATABASE CLOSE` statement or the `SHUTDOWN` command for a PDB, a keystore in the `OPEN` state for the PDB remains open until a user who has the `SYSKM` administrative privilege manually closes it with the `ADMINISTER KEY MANAGEMENT SET KEYSTORE CLOSE` statement.

You can check if a TDE wallet or keystore is closed by querying the `STATUS` column of the `V$ENCRYPTION_WALLET` view.

The following data operations will fail if the TDE wallet or keystore is not accessible:

- `SELECT` data from an encrypted column

- INSERT data into on an encrypted column
- CREATE a table with encrypted columns
- CREATE an encrypted tablespace

8.1.3.2 Closing a TDE Wallet in United Mode

You can close password-protected TDE wallets, auto-login TDE wallets, and local auto-login TDE wallets in united mode.

In the case of an auto-login TDE wallet, which opens automatically when it is accessed, you must first move it to a new location where it cannot be automatically opened, then you must manually close it. You must do this if you are changing your configuration from an auto-login TDE wallet to a password-protected TDE wallet: you change the configuration to stop using the auto-login TDE wallet (by moving the auto-login TDE wallet to another location where it cannot be automatically opened), and then closing the auto-login TDE wallet.

1. Connect to the CDB root as a common user who has been granted the ADMINISTER KEY MANAGEMENT or SYSKM privilege.
2. Use the ADMINISTER KEY MANAGEMENT statement to close the TDE wallet.
   - For a password-protected TDE wallet, use the following syntax if you are in the CDB root:

     ```
     ADMINISTER KEY MANAGEMENT SET | FORCE KEYSTORE CLOSE
     [IDENTIFIED BY [EXTERNAL STORE | TDE_wallet_password]]
     [CONTAINER = ALL | CURRENT];
     ```

     Use the SET clause to close the TDE wallet without force. If there is a dependent TDE wallet that is open (for example, an isolated mode PDB TDE wallet and you are trying to close the CDB root TDE wallet), then an ORA-46692 cannot close wallet error appears. If this happens, then use the FORCE clause instead of SET to temporarily close the dependent TDE wallet during the close operation. The STATUS column of the V$ENCRYPTION_WALLET view shows if a TDE wallet is open.

     If you are in the united mode PDB, then either omit the CONTAINER clause or set it to CURRENT.

   - For an auto-login or local auto-login TDE wallet, use this syntax if you are in the CDB root:

     ```
     ADMINISTER KEY MANAGEMENT SET KEYSTORE CLOSE
     [CONTAINER =  ALL | CURRENT];
     ```

Closing a TDE wallet disables all of the encryption and decryption operations. Any attempt to encrypt or decrypt data or access encrypted data results in an error.

8.1.3.3 Closing an External Keystore in United Mode

To close an external keystore, you must use the ADMINISTER KEY MANAGEMENT statement with the SET KEYSTORE CLOSE clause.

1. Connect to the CDB root as a common user who has been granted the ADMINISTER KEY MANAGEMENT or SYSKM privilege.
2. Use the ADMINISTER KEY MANAGEMENT statement to close the external keystore.

   ```
   ADMINISTER KEY MANAGEMENT SET KEYSTORE CLOSE
   IDENTIFIED BY [EXTERNAL STORE | "external_key_manager_password"]
   [CONTAINER = ALL | CURRENT];
   ```

For example:

```
ADMINISTER KEY MANAGEMENT SET KEYSTORE CLOSE
IDENTIFIED BY "external_key_manager_password"
CONTAINER = ALL;
```

If an `ORA-46692 cannot close wallet` error appears, then check if any isolated mode keystores are open. To find the status of a keystore, query the `STATUS` column of the `V$ENCRYPTION_WALLET` view.

Closing a keystore disables all of the encryption and decryption operations. Any attempt to encrypt or decrypt data or access encrypted data results in an error.

8.1.4 Creating TDE Master Encryption Keys for Later Use in United Mode

You can create a TDE master encryption key that can be activated at a later date.

8.1.4.1 About Creating a TDE Master Encryption Key for Later Use

The `CREATE KEY` clause of the `ADMINISTER KEY MANAGEMENT` statement can create a TDE master encryption key to be activated at a later date.

You then can activate this key on the same database or export it to another database and activate it there.

This method of TDE master encryption key creation is useful in a multitenant environment when you must re-create the TDE master encryption keys. The `CREATE KEY` clause enables you to use a single SQL statement to generate a new TDE master encryption key for all of the PDBs within a multitenant environment. The creation time of the new TDE master encryption key is later than the activation of the TDE master encryption key that is currently in use. Hence, the creation time can serve as a reminder to all of the PDBs to activate the most recently created TDE master encryption key as soon as possible.

8.1.4.2 Creating a TDE Master Encryption Key for Later Use in United Mode

A TDE wallet must be opened before you can create a TDE master encryption key for use later on in united mode.

1. Connect to the CDB root as a common user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.
2. Create the TDE master encryption key by using the following syntax:

```
ADMINISTER KEY MANAGEMENT CREATE KEY [USING TAG 'tag']
[FORCE KEYSTORE]
IDENTIFIED BY EXTERNAL STORE | keystore_password
WITH BACKUP [USING 'backup_identifier']
[CONTAINER = ALL | CURRENT];
```

In this specification:

- `FORCE KEYSTORE` temporarily opens the password-protected TDE wallet for this operation. You must open the TDE wallet for this operation.
- *`tag`* is the associated attribute and information that you define. Enclose this setting in single quotation marks (' ').
- `IDENTIFIED BY` can be one of the following settings:

  - EXTERNAL STORE uses the keystore password stored in the external store to perform the keystore operation.
  - *TDE_wallet_password* is the mandatory TDE wallet password that you used when you created the original TDE wallet. It is case sensitive.

- WITH BACKUP backs up the TDE master encryption key in the same location as the key, as identified by the WRL_PARAMETER column of the V$ENCRYPTION_WALLET view. To find the key locations for all of the database instances, query the GV$ENCRYPTION_WALLET view.

  You must back up password-based TDE wallets. You do not need to back up auto-login or local auto-login TDE wallets. Optionally, include the USING *backup_identifier* clause to add a description of the backup. Enclose *backup_identifier* in single quotation marks (' ').

- CONTAINER: In the CDB root, set CONTAINER to either ALL or CURRENT. In a PDB, set it to CURRENT. In both cases, omitting CONTAINER defaults to CURRENT.

For example:

```
ADMINISTER KEY MANAGEMENT CREATE KEY
FORCE KEYSTORE
IDENTIFIED BY keystore_password
WITH BACKUP
CONTAINER = CURRENT;
```

3. If necessary, activate the TDE master encryption key.

   a. Find the key ID.

   ```
   SELECT KEY_ID FROM V$ENCRYPTION_KEYS;

   KEY_ID
   ------------------------------------------------
   AWsHwVYC2U+Nv3RVphn/yAIAAAAAAAAAAAAAAAAAAAAAAAAAAAA
   ```

   b. Use this key ID to activate the key.

   ```
   ADMINISTER KEY MANAGEMENT USE KEY
   'AWsHwVYC2U+Nv3RVphn/yAIAAAAAAAAAAAAAAAAAAAAAAAAAAAA'
   USING TAG 'quarter:second;description:Activate Key on standby'
   IDENTIFIED BY password
   WITH BACKUP;
   ```

8.1.5 Example: Creating a Master Encryption Key in All PDBs

You can use the ADMINISTER KEY MANAGEMENT CREATE KEY USING TAG statement to create a TDE master encryption key in all PDBs.

Example 8-1 shows how to create a master encryption key in all of the PDBs in a multitenant environment. It uses the FORCE KEYSTORE clause in the event that the auto-login keystore in the CDB root is open. The password is stored externally, so the EXTERNAL STORE setting is used for the IDENTIFIED BY clause. After you run this statement, a master encryption key is created in each PDB. You can find the identifiers for these keys as follows:

- Log in to the PDB and then query the TAG column of the V$ENCRYPTION_KEYS view.
- Log in to the CDB root and then query the INST_ID and TAG columns of the GV$ENCRYPTION_KEYS view.

You also can check the CREATION_TIME column of these views to find the most recently created key, which would be the key that you created from this statement. After you create the keys, you can individually activate the keys in each of the PDBs.

Example 8-1 Creating a Master Encryption Key in All of the PDBs

```
ADMINISTER KEY MANAGEMENT CREATE KEY USING TAG
'scope:all pdbs;description:Create Key for ALL PDBS'
FORCE KEYSTORE IDENTIFIED BY EXTERNAL STORE
WITH BACKUP
CONTAINER = ALL;

keystore altered.
```

8.1.6 Activating TDE Master Encryption Keys in United Mode

After you activate a TDE master encryption key, it can be used.

8.1.6.1 About Activating TDE Master Encryption Keys

You can activate a previously created or imported TDE master encryption key by using the USE KEY clause of ADMINISTER KEY MANAGEMENT.

After you activate the master encryption key, it is used to encrypt all data encryption keys in your database. The key will be used to protect all of the column keys and all of the tablespace encryption keys. If you have deployed a logical standby database, then you must export the TDE master encryption keys after recreating them, and then import them into the standby database. You can have the TDE master encryption key in use on both the primary and the standby databases. To do so, you must activate the TDE master encryption key after you import it to the logical standby database.

8.1.6.2 Activating a TDE Master Encryption Key in United Mode

To activate a TDE master encryption key in united mode, you must open the keystore and use ADMINISTER KEY MANAGEMENT with the USE KEY clause.

1. Connect to the CDB root as a common user who has been granted the ADMINISTER KEY MANAGEMENT or SYSKM privilege.
2. Query the ORIGIN and KEY_ID columns of the V$ENCRYPTION_KEYS view to find the key identifier.

   For example:

   ```
   SELECT ORIGIN, KEY_ID FROM V$ENCRYPTION_KEYS;

   ORIGIN  KEY_ID
   ------  ----------------------------------------------
   LOCAL   ARaHD762tUkkvyLgPzAi6hMAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
   ```

3. Use this key identifier to activate the TDE master encryption key by using the following syntax:

   ```
   ADMINISTER KEY MANAGEMENT USE KEY 'key_identifier_from_V$ENCRYPTION_KEYS'
   [USING TAG 'tag']
   [FORCE KEYSTORE]
   IDENTIFIED BY [EXTERNAL STORE | keystore_password]
   WITH BACKUP [USING 'backup_identifier'];
   ```

   In this specification:

- `FORCE KEYSTORE` temporarily opens the password-protected TDE wallet for this operation. You must open the TDE wallet for this operation.
- `WITH BACKUP` backs up the wallet in the same location as original wallet, as identified by *`WALLET_ROOT`*`/tde`. To find the key locations for all of the database instances, query the `V$ENCRYPTION_WALLET` or `GV$ENCRYPTION_WALLET` view.

  The `WITH BACKUP` clause is mandatory for all `ADMINISTER KEY MANAGEMENT` statements that modify the wallet. Optionally, include the `USING` *`backup_identifier`* clause to add a description of the backup. Enclose `backup_identifier` in single quotation marks (`' '`).

For example:

```
ADMINISTER KEY MANAGEMENT USE KEY
'ARaHD762tUkkvyLgPzAi6hMAAAAAAAAAAAAAAAAAAAAAAAAAAAAA'
FORCE KEYSTORE
IDENTIFIED BY EXTERNAL STORE
WITH BACKUP;
```

8.1.6.3 Example: Activating a TDE Master Encryption Key

You can use the `ADMINISTER KEY MANAGEMENT` SQL statement to activate a TDE master encryption key.

Example 8-2 shows how to activate a previously imported TDE master encryption key and then update its tag. This key is activated with the current database time stamp and time zone.

Example 8-2 Activating a TDE Master Encryption Key

```
ADMINISTER KEY MANAGEMENT USE KEY
'ARaHD762tUkkvyLgPzAi6hMAAAAAAAAAAAAAAAAAAAAAAAAAAAAA'
USING TAG 'quarter:second;description:Activate Key on standby'
IDENTIFIED BY password WITH BACKUP;

keystore altered.
```

In this version of the same operation, the `FORCE KEYSTORE` clause is added in the event that the auto-login keystore is in use, or if the keystore is closed. The password of the keystore is stored externally, so the `EXTERNAL STORE` setting is used for the `IDENTIFIED BY` clause.

```
ADMINISTER KEY MANAGEMENT USE KEY
'ARaHD762tUkkvyLgPzAi6hMAAAAAAAAAAAAAAAAAAAAAAAAAAAAA'
USING TAG 'quarter:second;description:Activate Key on standby'
FORCE KEYSTORE IDENTIFIED BY EXTERNAL STORE WITH BACKUP;

keystore altered.
```

8.1.7 Creating User-Defined TDE Master Encryption Keys

You can create a user-defined TDE master encryption key outside the database by generating a TDE master encryption key ID.

8.1.7.1 About User-Defined TDE Master Encryption Keys

A TDE master encryption key that is outside the database has its own user-generated ID, which tracks the use of the TDE master encryption key.

You can use the `ADMINISTER KEY MANAGEMENT` to create and set user-defined TDE master encryption key IDs. After you generate the TDE master encryption key, you can bring this key

into the database. Optionally, you can specify the TDE master encryption key ID in various `ADMINISTER KEY MANAGEMENT` statements.

This type of configuration benefits Oracle Fusion SaaS Cloud environments in that it enables you to generate a TDE master encryption key this complies with your site's requirements. This key that you generate supports the current encryption algorithms and can be used for TDE wallets.

After you generate the TDE master encryption key ID, you can encrypt your data as you normally would.

The TDE master encryption key and its corresponding ID will not be captured by any auditing logs.

8.1.7.2 Creating a User-Defined TDE Master Encryption Key in United Mode

To create a user-defined TDE master encryption key, use the `ADMINISTER KEY MANAGEMENT` statement with the `SET | CREATE [ENCRYPTION] KEY` clause.

1. Connect to the CDB root as a common user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.
2. Create the user-defined TDE master encryption key by using the following syntax:

```
ADMINISTER KEY MANAGEMENT SET | CREATE [ENCRYPTION] KEY
'mkid:mk | mk'
[USING ALGORITHM 'algorithm']
[FORCE KEYSTORE]
[USING TAG 'tag_name']
IDENTIFIED BY [EXTERNAL STORE | keystore_password]
WITH BACKUP [USING 'backup_identifier'];
[CONTAINER = CURRENT];
```

In this specification:

- `SET | CREATE` : Enter `SET` if you want to create the master and activate the TDE master encryption key now, or enter `CREATE` if you want to create the key for later use, without activating it yet. For `SET KEY` and `USE KEY`, you must be connected to the database that is supposed to use that key (CDB root container, united or isolated PDB). For `CREATE KEY`, you can be connected any united PDB, or the CDB root container, or the specific isolated PDB that is supposed to use that key later.
- *`mkid`* and *`mk`*:
  - *`mkid`*, the TDE master encryption key ID, is an optional 16-byte hex-encoded value that you can specify or have Oracle Database generate.
  - *`mk`*, the TDE master encryption key, is a 32-byte hex-encoded value.

  If you omit the *`mkid`* value but include the *`mk`*, then Oracle Database generates the *`mkid`* for the *`mk`*.
- `USING ALGORITHM`: Specify one of the following supported algorithms:
  - `AES256`
  - `ARIA256`

  If you omit the algorithm, then the default, `AES256`, is used.
- `FORCE KEYSTORE` temporarily opens the password-protected TDE wallet for this operation. You must open the TDE wallet for this operation.

The following example includes a user-created TDE master encryption key but no TDE master encryption key ID, so that the TDE master encryption key is generated:

```
ADMINISTER KEY MANAGEMENT SET ENCRYPTION KEY
'3D432109DF88967A541967062A6F4E460E892318E307F017BA048707B402493C'
USING ALGORITHM 'ARIA256'
FORCE KEYSTORE
IDENTIFIED BY keystore_password WITH BACKUP;
```

The next example creates user-defined keys for both the master encryption ID and the TDE master encryption key. It omits the algorithm specification, so the default algorithm `AES256` is used.

```
ADMINISTER KEY MANAGEMENT CREATE ENCRYPTION KEY
'10203040506070801112131415161718:3D432109DF88967A541967062A6F4E460E892318E307F017BA0
48707B402493C'
IDENTIFIED BY keystore_password WITH BACKUP;
```

The next scenario, where TDE master encryption key and key-ID are generated outside of the database, shows how to support separation of duties. A key administrator inserts the key with a `key_ID` into the wallet by using the `ADMINISTER KEY MANAGEMENT CREATE [ENCRYPTION] KEY` statement. Then, the key administrator sends the `key_ID` to the database administrator, who then activates the key by using the `ADMINISTER KEY MANAGEMENT USE KEY` clause.

a. Create the `key-ID`, converting characters to upper case. For example, using OpenSSL:

```
$ openssl rand 16 | xxd -u -p
D20765EB721AF44D054B30FE87F8E49A
```

b. Create the TDE master encryption key. For example, using OpenSSL:

```
$ openssl rand -hex 32
5a089c8ea6ee21cba774f61e58d102665df4a979a34757836b3d066a3eb3db11
```

c. Create (but do not activate) a default AES256 master encryption key by inserting both the random `key-ID` and the key itself, separated by a `:`, into the wallet.

```
ADMINISTER KEY MANAGEMENT CREATE ENCRYPTION KEY
'D20765EB721AF44D054B30FE87F8E49A:5a089c8ea6ee21cba774f61e58d102665df4a979a347578
36b3d066a3eb3db11'
IDENTIFIED BY [EXTERNAL STORE | keystore_password]
WITH BACKUP [USING 'backup_identifier'];
```

d. The database administrator can retrieve the BASE64 key-ID from the database with the following `SELECT` statement:

```
SELECT ' ADMINISTER KEY MANAGEMENT USE KEY '''||KEY_ID||'''
USING TAG '''||SYS_CONTEXT('USERENV', 'CON_NAME')||' '||TO_CHAR (SYS_EXTRACT_UTC
(SYSTIMESTAMP),
'YYYY-MM-DD HH24:MI:SS"Z"')||'''
FORCE KEYSTORE
IDENTIFIED BY EXTERNAL STORE WITH BACKUP;'
AS "USE KEY COMMAND" FROM V$ENCRYPTION_KEYS WHERE TAG IS NULL;
```

Related Topics

- Supported Encryption and Integrity Algorithms
  Oracle supports the AES, ARIA and DES algorithms.

8.1.8 Rekeying the TDE Master Encryption Key in United Mode

You can use the `ADMINISTER KEY MANAGEMENT` statement with the `SET KEY` clause to rekey a TDE master encryption key.

1. Connect to the CDB root as a common user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.
2. If you are rekeying the TDE master encryption key for a keystore that has auto login enabled, then ensure that both the auto login keystore, identified by the `.sso` file, and the encryption keystore, identified by the `.p12` file, are present.

   You can find the location of these files by querying the `WRL_PARAMETER` column of the `V$ENCRYPTION_WALLET` view. To find the `WRL_PARAMETER` values for all of the database instances, query the `GV$ENCRYPTION_WALLET` view.
3. Rekey the TDE master encryption key by using the following syntax:

   ```
   ADMINISTER KEY MANAGEMENT SET [ENCRYPTION] KEY
   [FORCE KEYSTORE]
   [USING TAG 'tag_name']
   IDENTIFIED BY [EXTERNAL STORE | keystore_password]
   [WITH BACKUP [USING 'backup_identifier']]
   [CONTAINER = ALL | CURRENT];
   ```

   In this specification:

   - *tag* is the associated attributes and information that you define. Enclose this setting in single quotation marks (`' '`).
   - `FORCE KEYSTORE` temporarily opens the password-protected TDE wallet for this operation. You must open the TDE wallet for this operation.
   - `IDENTIFIED BY` can be one of the following settings:
     - `EXTERNAL STORE` uses the keystore password stored in the external store to perform the keystore operation.
     - *keystore_password* is the password that was created for this keystore.
   - `CONTAINER`: In the CDB root, set `CONTAINER` to either `ALL` or `CURRENT`. In a PDB, set it to `CURRENT`. In both cases, omitting `CONTAINER` defaults to `CURRENT`.

   For example:

   ```
   ADMINISTER KEY MANAGEMENT SET KEY
   FORCE KEYSTORE
   IDENTIFIED BY keystore_password
   WITH BACKUP USING 'emp_key_backup'
   CONTAINER = CURRENT;

   keystore altered.
   ```

Related Topics

- About Rekeying the TDE Master Encryption Key
  Oracle Database uses a unified TDE Master Encryption Key for both TDE column encryption and TDE tablespace encryption.

8.1.9 Finding the TDE Master Encryption Key That Is in Use in United Mode

A TDE master encryption key that is in use is the key that was activated most recently for the database.

In united mode, the TDE master encryption key in use of the PDB is the one that was activated most recently for that PDB.

- To find the TDE master encryption key that is in use, query the `V$ENCRYPTION_KEYS` dynamic view.

  For example:

```
SELECT KEY_ID
FROM V$ENCRYPTION_KEYS
WHERE ACTIVATION_TIME = (SELECT MAX(ACTIVATION_TIME)
       FROM V$ENCRYPTION_KEYS
       WHERE ACTIVATING_PDBID = SYS_CONTEXT('USERENV', 'CON_ID'));
```

8.1.10 Creating a Custom Attribute Tag in United Mode

To create a custom attribute tag in united mode, you must use the `SET TAG` clause of the `ADMINISTER KEY MANAGEMENT` statement.

1. Connect to the CDB root as a common user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.
2. If necessary, query the `TAG` column of the `V$ENCRYPTION_KEY` dynamic view to find a listing of existing tags for the TDE master encryption keys.

   When you create a new tag for a TDE master encryption key, it overwrites the existing tag for that TDE master encryption key.
3. Create the custom attribute tag by using the following syntax:

```
ADMINISTER KEY MANAGEMENT SET TAG 'tag'
FOR 'master_key_identifier'
[FORCE KEYSTORE]
IDENTIFIED BY [EXTERNAL STORE | keystore_password]
WITH BACKUP [USING 'backup_identifier'];
```

   In this specification

   - *tag* is the associated attributes or information that you define. Enclose this information in single quotation marks (`' '`).
   - *master_key_identifier* identifies the TDE master encryption key for which the *tag* is set. To find a list of TDE master encryption key identifiers, query the `KEY_ID` column of the `V$ENCRYPTION_KEYS` dynamic view.
   - `FORCE KEYSTORE` temporarily opens the password-protected TDE wallet for this operation. You must open the TDE wallet for this operation.
   - `IDENTIFIED BY` can be one of the following settings:
     - `EXTERNAL STORE` uses the keystore password stored in the external store to perform the keystore operation.
     - *keystore_password* is the password that was created for this keystore.
   - *backup_identifier* defines the tag values. Enclose this setting in single quotation marks (`' '`) and separate each value with a colon.

For example, to create a tag that uses two values, one to capture a specific session ID and the second to capture a specific terminal ID:

```
ADMINISTER KEY MANAGEMENT SET ENCRYPTION KEY
USING TAG 'sessionid=3205062574:terminal=xcvt'
IDENTIFIED BY keystore_password
WITH BACKUP;

keystore altered.
```

Both the session ID (`3205062574`) and terminal ID (`xcvt`) can derive their values by using either the `SYS_CONTEXT` function with the `USERENV` namespace, or by using the `USERENV` function.

Related Topics

- About Creating Custom Attribute Tags
  Attribute tags enable you to monitor specific activities users perform, such as accessing a particular terminal ID.

8.1.11 Moving TDE Master Encryption Keys into a New Keystore in United Mode

You can move an existing TDE master encryption key into a new keystore from an existing password-protected keystore.

8.1.11.1 About Moving TDE Master Encryption Keys into a New Keystore

You can move an unused (and safely archived) TDE master encryption key into a new keystore.

Use great caution when you decide to run `ADMINISTER KEY MANAGEMENT MOVE KEYS`. Even though this statement will not move an active master encryption key, it can still affect keys that are necessary for a range of database features. If you have deleted a key, then the data that was encrypted by that key is rendered permanently inaccessible (equivalent to deleting the data) for these features to be used. See Related Topics at the end of this topic for more information about features that are affected by deleted keystores.

Therefore, before you move the keystore, it is very important that you safely archive it. Delete the keystore only after a period of time has passed, to ensure that the keystore is no longer really useful.

To move a TDE master encryption key into a new keystore, you use the `ADMINISTER KEY MANAGEMENT MOVE KEYS` statement. This statement does not move the active TDE master encryption key (that is, the key that is currently in use at the time that `ADMINISTER KEY MANAGEMENT MOVE KEYS` is issued) because the database is currently using it.

If you mistakenly use the `ADMINISTER KEY MANAGEMENT MOVE KEYS` statement instead of `ADMINISTER KEY MANAGEMENT MERGE KEYSTORE` when you are configuring a new TDE keystore (for example, when you are changing the TDE keystore configuration from one where the TDE wallet is located in the operating system's file system to one where the TDE wallet is located in Oracle Automatic Storage Management (Oracle ASM)), then the following symptoms may help you to identify the TDE misconfiguration that was introduced by the use of the wrong key management command:

- When you open the TDE keystore that was the target of the earlier `ADMINISTER KEY MANAGEMENT MOVE KEYS` operation, an `ORA-28374: typed master key not found in`

`wallet` error is seen, because the active TDE master encryption key was not moved to that keystore.

- The value shown in the `STATUS` column of the `V$ENCRYPTION_WALLET` view is `OPEN_NO_MASTER_KEY` after you open the new keystore. The `OPEN_NO_MASTER_KEY` status is expected, because the new TDE keystore that was mistakenly populated by means of the `ADMINISTER KEY MANAGEMENT MOVE KEYS` statement does not contain the active TDE master encryption key.

Related Topics

- Dangers of Deleting TDE Wallets
  Oracle strongly recommends that you do not delete TDE wallets.
- Features That Are Affected by Deleted Keystores
  Some features can be adversely affected if a keystore is deleted and a TDE master encryption key residing in that keystore is later needed.

8.1.11.2 Moving a TDE Master Encryption Key into a New Keystore in United Mode

In united mode, you can move an existing TDE master encryption key into a new keystore from an existing password-based TDE wallet.

This feature enables you to delete unused keystores. Before you move the keystore, ensure that it is safely archived. After you move the encryption key to a new keystore, and when you are sure that the old keystore is no longer needed, you then can delete the old keystore.

1. Connect to the CDB root as a common user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.
2. Query the `KEY_ID` column of the `V$ENCRYPTION_KEYS` view to find the key identifier of the keystore to which you want to move the keys.

   For example:

```
SELECT CREATION_TIME, KEY_ID FROM V$ENCRYPTION_KEYS;

CREATION TIME
---------------------------------------------------
23-SEP-19 08.55.12.956170 PM +00:00

KEY_ID
---------------------------------------------------
ARaHD762tUkkvyLgPzAi6hMAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```

3. Move the key into a new keystore by using the following syntax:

```
ADMINISTER KEY MANAGEMENT
MOVE [ENCRYPTION] KEYS
TO NEW KEYSTORE 'keystore_location1'
IDENTIFIED BY keystore1_password
FROM [FORCE] KEYSTORE
IDENTIFIED BY keystore_password
[WITH IDENTIFIER IN
{ 'key_identifier' [, 'key_identifier' ]... | ( subquery ) } ]
[WITH BACKUP [USING 'backup_identifier']];
```

   In this specification:

   - *keystore_location1* is the path to the `wallet` directory that will store the new keystore `.p12` file. By default, this directory is in `$ORACLE_BASE/admin/`*db_unique_name*`/wallet`.

- *keystore1_password* is the password for the keystore from which the new keystore is moved.
- FORCE temporarily opens the keystore for this operation.
- *keystore_password* is the password for the keystore from which the key is moving.
- *subquery* can be used to find the exact key identifier that you want.
- *backup_identifier* is an optional description of the backup. Enclose *backup_identifier* in single quotation marks (' ').

For example:

```
ADMINISTER KEY MANAGEMENT MOVE KEYS
TO NEW KEYSTORE '$ORACLE_BASE/admin/orcl/wallet'
IDENTIFIED BY keystore_password
FROM FORCE KEYSTORE
IDENTIFIED BY keystore_password
WITH IDENTIFIER IN
(SELECT KEY_ID FROM V$ENCRYPTION_KEYS WHERE ROWNUM < 2)
WITH BACKUP;
```

4. To delete the old keystore, go to the wallet directory and do the following:
   a. Back up the .p12 file containing the keystore that you want to delete.
   b. Manually delete the .p12 file containing the keystore.

   To find the location of the keystore, open the keystore, and then query the WRL_PARAMETER column of the V$ENCRYPTION_WALLET view.

Related Topics

- Dangers of Deleting TDE Wallets
  Oracle strongly recommends that you do not delete TDE wallets.

8.1.12 Automatically Removing Inactive TDE Master Encryption Keys in United Mode

In united mode, the REMOVE_INACTIVE_STANDBY_TDE_MASTER_KEY initialization parameter can configure the automatic removal of inactive TDE master encryption keys.

1. Connect to the CDB root as a common user who has been granted the ADMINISTER KEY MANAGEMENT or SYSKM privilege.
2. Locate the initialization parameter file for the database.

   By default, the initialization parameter fileis located in the $ORACLE_HOME/dbs directory.
3. Edit the initialization parameter file to include the REMOVE_INACTIVE_STANDBY_TDE_MASTER_KEY initialization parameter.

   For example:

   ```
   remove_inactive_standby_tde_master_key = true
   ```

   Setting this parameter to TRUE enables the automatic removal of inactive TDE master encryption keys; setting it to FALSE disables the automatic removal.

8.1.13 Isolating a Pluggable Database Keystore

Isolating a PDB keystore moves the master encryption key from the CDB root keystore into an isolated mode keystore in the a PDB.

This process enables the keystore to be managed as a separate keystore in isolated mode. This way, an administrator who has been locally granted the `ADMINISTER KEY MANAGEMENT` privilege for the PDB can manage the keystore.

1. Connect to the CDB root as a common user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.
2. Move the keys from the keystore of the CDB root into the isolated mode keystore of the PDB by using the following syntax:

```
ADMINISTER KEY MANAGEMENT [FORCE] ISOLATE KEYSTORE
IDENTIFIED BY isolated_keystore_password
FROM ROOT KEYSTORE
[FORCE KEYSTORE]
IDENTIFIED BY
[EXTERNAL STORE | united_keystore_password]
[WITH BACKUP [USING backup_id]];
```

   In this specification:

   - `FORCE` is used when a clone of the PDB is using the master encryption key that is being isolated. The `ADMINISTER KEY MANAGEMENT` statement then copies (rather than moves) the keys from the wallet of the CDB root into the isolated mode PDB.
   - `FORCE KEYSTORE` temporarily opens the password-protected TDE wallet for this operation if an auto-login TDE wallet is open (and in use) or if the TDE wallet is closed.
   - *`united_keystore_password`*: Knowledge of this password does not enable the user who performs the `ISOLATE KEYSTORE` operation privileges to perform `ADMINISTER KEY MANAGEMENT UNITE KEYSTORE` operations on the CDB root. This password is the same as the keystore password in the CDB root.

   After the keystore of a CDB root has been united with that of a PDB, all of the previously active (historical) master encryption keys that were associated with the CDB are moved to the keystore of the PDB.
3. Confirm that the united mode PDB is now an isolated mode PDB.

```
SELECT KEYSTORE_MODE FROM V$ENCRYPTION_WALLET;
```

   The output should be `ISOLATED`.

After the united mode PDB has been converted to an isolated mode PDB, you can change the password of the keystore.

8.2 Administering Transparent Data Encryption in United Mode

You can perform general administrative tasks with Transparent Data Encryption in united mode.

8.2.1 Moving PDBs from One CDB to Another in United Mode

You can clone or relocate encrypted PDBs within the same container database, or across container databases.

If you are trying to move a non-CDB or a PDB in which the `SYSTEM`, `SYSAUX`, `UNDO`, or `TEMP` tablespace is encrypted, and using the manual export or import of keys, then you must first import the keys for the non-CDB or PDB in the target database's `CDB$ROOT` before you create the PDB. Import of the keys are again required inside the PDB to associate the keys to the PDB.

- Clone or relocate the non-CDB or PDB using the following syntax:

```
CREATE|RELOCATE PLUGGABLE DATABASE database_name KEYSTORE
IDENTIFIED BY EXTERNAL STORE|target_keystore_password [NO REKEY];
```

Related Topics

- *Oracle Multitenant Administrator's Guide*

8.2.2 Unplugging and Plugging a PDB with Encrypted Data in a CDB in United Mode

In united mode, for a PDB that has encrypted data, you can plug it into a CDB. Conversely, you can unplug this PDB from the CDB.

8.2.2.1 Unplugging a PDB That Has Encrypted Data in United Mode

In united mode, you can unplug a PDB with encrypted data and export it into an XML file or an archive file.

You can check if a PDB has been unplugged by querying the `STATUS` column of the `DBA_PDBS` data dictionary view.

1. Connect to the CDB root as a common user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.
2. Query the `V$ENCRYPTION_WALLET` dynamic view to ensure that the keystore is open.
3. Use the `ENCRYPT USING transport_secret` clause in the `ALTER PLUGGABLE DATABASE` statement when you unplug the PDB.

   This process extracts the master encryption keys that belong to that PDB from the open wallet, and encrypts those keys with the `transport_secret` clause.

   You must use this clause if the PDB has encrypted data. Otherwise, an `ORA-46680: master keys of the container database must be exported` error is returned.

   - For example, to export the PDB data into an XML file:

     ```
     ALTER PLUGGABLE DATABASE CDB1_PDB2
     UNPLUG INTO '/tmp/cdb1_pdb2.xml'
     ENCRYPT USING transport_secret;
     ```

   - To export the PDB data into an archive file:

     ```
     ALTER PLUGGABLE DATABASE CDB1_PDB2
     UNPLUG INTO '/tmp/cdb1_pdb2.pdb'
     ENCRYPT USING transport_secret;
     ```

Related Topics

- Opening the TDE Wallet in a United Mode PDB
  To open a TDE wallet in united mode, you must use the `ADMINISTER KEY MANAGEMENT` statement with the `SET KEYSTORE OPEN` clause.
- *Oracle Multitenant Administrator's Guide*
- *Oracle Database SQL Language Reference*

8.2.2.2 Plugging a PDB That Has Encrypted Data into a CDB in United Mode

To plug a PDB that has encrypted data into a CDB, you first plug in the PDB and then you create a master encryption key for the PDB.

After you plug the PDB into the target CDB, and you must create a master encryption key that is unique to this plugged-in PDB. This is because the plugged-in PDB initially uses the key that was extracted from the wallet of the source PDB. When you plug an unplugged PDB into another CDB, the key version is set to `0` because this operation invalidates the history of the previous keys. You can check the key version by querying the `KEY_VERSION` column of the `V$ENCRYPTED_TABLESPACES` dynamic view. Similarly, if a control file is lost and recreated, then the previous history of the keys is reset to `0`. You can check if a PDB has already been plugged in by querying the `STATUS` column of the `DBA_PDBS` data dictionary view.

1. Connect to the CDB root as a common user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.
2. Create the PDB by plugging the unplugged PDB into the CDB.

   To perform this operation for united mode, include the `DECRYPT USING` *`transport_secret`* clause.

   You must use this clause if the XML or archive file for the PDB has encrypted data. Otherwise, an `ORA-46680: master keys of the container database must be exported` error is returned.

   - For example, if you had exported the PDB data into an XML file:

     ```
     CREATE PLUGGABLE DATABASE CDB1_PDB2
     USING '/tmp/cdb1_pdb2.xml'
     KEYSTORE IDENTIFIED BY EXTERNAL STORE|TDE_wallet_password
     DECRYPT USING transport_secret;
     ```

   - If you had exported the PDB into an archive file:

     ```
     CREATE PLUGGABLE DATABASE CDB1_PDB2
     USING '/tmp/cdb1_pdb2.pdb'
     KEYSTORE IDENTIFIED BY EXTERNAL STORE|TDE_wallet_password
     DECRYPT USING transport_secret;
     ```

   During the open operation of the PDB after the plug operation, Oracle Database determines if the PDB has encrypted data. If so, it opens the PDB in the `RESTRICTED` mode.

   If you want to create the PDB by cloning another PDB or from a non-CDB, and if the source database has encrypted data or a TDE master encryption key that has been set, then you must provide the TDE wallet password of the target TDE wallet by including the `KEYSTORE IDENTIFIED BY` *`TDE_wallet_password`* clause in the `CREATE PLUGGABLE DATABASE ... FROM` SQL statement. You must provide this password even if the target database is using an auto-login TDE wallet. You can find if the source database has encrypted data or a TDE master encryption key set in the TDE wallet by querying the `V$ENCRYPTION_KEYS` dynamic view
3. Open the PDB.

```
ALTER PLUGGABLE DATABASE pdb_name OPEN;
```

4. Open the TDE wallet in the CDB root by using one of the following methods:
   - If the TDE wallet of the CDB is not open, open it for the container and all open PDBs by using the following syntax:

     ```
     ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN [FORCE KEYSTORE]
     IDENTIFIED BY EXTERNAL STORE|KEYSTORE_PASSWORD CONTAINER = ALL;
     ```

   - If the TDE wallet of the CDB is open, connect to the plugged-in PDB and then open the keystore by using the following syntax:

     ```
     ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN [FORCE KEYSTORE]
     IDENTIFIED BY EXTERNAL STORE|KEYSTORE_PASSWORD [CONTAINER = CURRENT];
     ```

5. Optionally, open the keystore in the PDB.
6. In the plugged-in PDB, set the TDE master encryption key for the PDB by using the following syntax:

   ```
   ADMINISTER KEY MANAGEMENT SET KEY
   [FORCE KEYSTORE]
   IDENTIFIED BY EXTERNAL STORE|TDE_wallet_password
   WITH BACKUP [USING 'backup_identifier'];
   ```

Related Topics

- Opening the TDE Wallet in a United Mode PDB
  To open a TDE wallet in united mode, you must use the `ADMINISTER KEY MANAGEMENT` statement with the `SET KEYSTORE OPEN` clause.
- Setting the TDE Master Encryption Key in the United Mode TDE Wallet
  To set the TDE master encryption key in the TDE wallet when the PDB is configured in united mode, use the `ADMINISTER KEY MANAGEMENT` statement with the `SET KEY` clause.

8.2.2.3 Unplugging a PDB That Has Master Encryption Keys Stored in an External Keystore in United Mode

You can unplug a PDB from one CDB that has been configured with an external keystore and then plug it into another CDB also configured with an external keystore.

1. Unplug the PDB.

   You can check if a PDB has already been unplugged by querying the `STATUS` column of the `DBA_PDBS` data dictionary view.

2. Move the master encryption keys of the unplugged PDB in the external keystore that was used at the source CDB to the external keystore that is in use at the destination CDB.

   Refer to the documentation for the external keystore for information about moving master encryption keys between external keystores.

Related Topics

- *Oracle Multitenant Administrator's Guide*

8.2.2.4 Plugging a PDB That Has Master Encryption Keys Stored in an External Keystore in United Mode

The `ADMINISTER KEY MANAGEMENT` statement can import a TDE master encryption key from an external keystore to a PDB that has been moved to another CDB.

1. Plug the unplugged PDB into the destination CDB that has been configured with the external keystore.

   You can check if a PDB has already been plugged in by querying the `STATUS` column of the `DBA_PDBS` data dictionary view.

   After the plug-in operation, the PDB that has been plugged in will be in restricted mode.
2. Ensure that the master encryption keys from the external keystore that has been configured with the source CDB are available in the external keystore of the destination CDB.
3. Connect to the plugged PDB as a user who was granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.
4. Open the master encryption key of the plugged PDB.

   For example:

   ```
   ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN
   IDENTIFIED BY keystore_passsword;
   ```
5. Import the external keystore master encryption key into the PDB.

   ```
   ADMINISTER KEY MANAGEMENT IMPORT ENCRYPTION KEYS
   WITH SECRET "OKV" FROM 'OKV'
   IDENTIFIED BY keystore_password;
   ```
6. Restart the PDB.

   ```
   ALTER PLUGGABLE DATABASE PDB1 CLOSE;
   ALTER PLUGGABLE DATABASE PDB1 OPEN;
   ```

Related Topics

- *Oracle Multitenant Administrator's Guide*

8.2.3 Managing Cloned PDBs with Encrypted Data in United Mode

In united mode, you can clone a PDB that has encrypted data in a CDB.

8.2.3.1 About Managing Cloned PDBs That Have Encrypted Data in United Mode

When you clone a PDB, you must make the master encryption key of the source PDB available to cloned PDB.

This allows a cloned PDB to operate on the encrypted data. To perform the clone, you do not need to export and import the keys because Oracle Database transports the keys for you even if the cloned PDB is in a remote CDB. However, you will need to provide the keystore password of the CDB where you are creating the clone.

If the PDBs have encrypted data, then you can perform remote clone operations on PDBs between CDBs, and relocate PDBs across CDBs.

8.2.3.2 Cloning a PDB with Encrypted Data in a CDB in United Mode

The `CREATE PLUGGABLE DATABASE` statement with the `KEYSTORE IDENTIFIED BY` clause can clone a PDB that has encrypted data.

1. Connect to the CDB root as a common user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.
2. Query the `STATUS` column of the `V$ENCRYPTION_WALLET` dynamic view to ensure that the keystore is open in the CDB root.
3. Log in to the PDB as a user who has been granted the `ADMINISTER KEY MANAGEMENT` and the `CREATE PLUGGABLE DATABASE` privileges.
4. Use the `CREATE PLUGGABLE DATABASE` statement with the `KEYSTORE IDENTIFIED BY` clause to clone the PDB.

   For example:

   ```
   CREATE PLUGGABLE DATABASE cdb1_pdb3
   FROM cdb1_pdb1
   FILE_NAME_CONVERT=('cdb1_pdb1', 'pdb3/cdb1_pdb3') KEYSTORE
   IDENTIFIED BY EXTERNAL STORE|keystore_password [NO REKEY];
   ```

   Replace *keystore_password* with the password of the keystore of the CDB where the `cdb1_pdb3` clone is created.

   By default, during a PDB clone or relocate operation, the data encryption keys are rekeyed, which implies a re-encryption of all encrypted tablespaces. This rekey operation can increase the time it takes to clone or relocate a large PDB. With the optional `NO REKEY` clause, the data encryption keys are not renewed, and encrypted tablespaces are not re-encrypted.

   After you create the cloned PDB, encrypted data is still accessible by the clone using the master encryption key of the original PDB. After a PDB is cloned, there may be user data in the encrypted tablespaces. This encrypted data is still accessible because the master encryption key of the source PDB is copied over to the destination PDB. Because the clone is a copy of the source PDB but will eventually follow its own course and have its own data and security policies, you should rekey the master encrytion key of the cloned PDB.
5. Rekey the master encryption key of the cloned PDB.

   For example:

   ```
   ADMINISTER KEY MANAGEMENT SET KEY
   FORCE KEYSTORE
   IDENTIFIED BY keystore_password
   WITH BACKUP USING 'emp_key_backup';
   ```

   In this example, `FORCE KEYSTORE` is included because the keystore must be open during the rekey operation.

   Before you rekey the master encryption key of the cloned PDB, the clone can still use master encryption keys that belong to the original PDB. However, these master encryption keys do not appear in the cloned PDB `V$` dynamic views. Rekeying the master encryption key ensures that the cloned PDB uses its own unique keys, which will be viewable in the `V$` views.

Related Topics

- Opening the TDE Wallet in a United Mode PDB
  To open a TDE wallet in united mode, you must use the `ADMINISTER KEY MANAGEMENT` statement with the `SET KEYSTORE OPEN` clause.

8.2.3.3 Remotely Clone an Encrypted PDB in United Mode

The `CREATE PLUGGABLE DATABASE` statement with the `KEYSTORE IDENTIFIED BY` clause can remotely clone a PDB that has encrypted data.

1. Connect to the CDB root as a common user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.
2. Query the `STATUS` column of the `V$ENCRYPTION_WALLET` dynamic view to ensure that the keystore is open in the CDB root.
3. In this root container of the target database, create a database link that connects to the root container of the source CDB.

   ```
   CREATE DATABASE LINK clone_link
   CONNECT TO C##REMOTE_CLONE_USER
   IDENTIFIED BY C##REMOTE_CLONE_USER_PASSWORD
   USING '//source_ip_address:port/DB_NAME';
   ```

4. Use the `CREATE PLUGGABLE DATABASE` statement with the `KEYSTORE IDENTIFIED BY` clause to perform the clone of the PDB.

   For example:

   ```
   CREATE PLUGGABLE DATABASE cdb1_pdb3
   FROM cbd_1_pdb1@clone_link
   FILE_NAME_CONVERT=('cdb1_pdb1', 'pdb3/cdb1_pdb3') KEYSTORE
   KEYSTORE IDENTIFIED BY EXTERNAL STORE|keystore_password;
   ```

   Replace *keystore_password* with the password of the keystore of the CDB where the `cdb1_pdb3` clone is created.

   After you create the cloned PDB, encrypted data is still accessible by the clone using the master encryption key of the original PDB. After a PDB is cloned, there may be user data in the encrypted tablespaces. This encrypted data is still accessible because the master encryption key of the source PDB is copied over to the destination PDB. Because the clone is a copy of the source PDB but will eventually follow its own course and have its own data and security policies, you should rekey the master encrytion key of the cloned PDB.

5. Rekey the master encryption key of the remotely cloned PDB.

   For example:

   ```
   ADMINISTER KEY MANAGEMENT SET KEY
   FORCE KEYSTORE
   IDENTIFIED BY keystore_password
   WITH BACKUP USING 'emp_key_backup';
   ```

   In this example, `FORCE KEYSTORE` is included because the keystore must be open during the rekey operation.

   Before you rekey the master encryption key of the cloned PDB, the clone can still use master encryption keys that belong to the original PDB. However, these master encryption keys do not appear in the cloned PDB `V$` dynamic views. Rekeying the master encryption

key ensures that the cloned PDB uses its own unique keys, which will be viewable in the V$ views.

Related Topics

- Opening the TDE Wallet in a United Mode PDB
  To open a TDE wallet in united mode, you must use the ADMINISTER KEY MANAGEMENT statement with the SET KEYSTORE OPEN clause.
- *Oracle Multitenant Administrator's Guide*
- *Oracle Multitenant Administrator's Guide*
- *Oracle Database SQL Language Reference*

8.2.3.4 Relocating an Encrypted PDB in United Mode

The CREATE PLUGGABLE DATABASE statement with the KEYSTORE IDENTIFIED BY clause can relocate a PDB with encrypted data across CDBs.

1. Connect to the CDB root as a common user who has been granted the ADMINISTER KEY MANAGEMENT or SYSKM privilege.
2. Query the STATUS column of the V$ENCRYPTION_WALLET dynamic view to ensure that the keystore is open in the CDB root.
3. Create a database link for the PDB that you want to clone.

   Use the CREATE DATABASE LINK SQL statement to create the database link. You must create the database link by following the database link prerequisites that are required for cloning a remote PDB.
4. Use the CREATE PLUGGABLE DATABASE statement with the KEYSTORE IDENTIFIED BY clause to relocate the PDB.

   For example:

   ```
   CREATE PLUGGABLE DATABASE cdb1_pdb3
   FROM cdb1_pdb1@clone_link RELOCATE [AVAILABILITY MAX]
   [FILE_NAME_CONVERT=('cdb1_pdb1', 'pdb3/cdb1_pdb3')]
   KEYSTORE IDENTIFIED BY EXTERNAL STORE|keystore_password;
   ```

   Replace *keystore_password* with the password of the keystore of the CDB where the cdb1_pdb3 clone is created.

   After you have relocated the PDB, the encrypted data is still accessible because the master encryption key of the source PDB is copied over to the destination PDB. The relocated PDB is a copy of the source PDB, but it will eventually follow its own course and have its own data and security policies. Therefore, you should rekey the master encrytion key of the cloned PDB.
5. Connect to the PDB as a user who has been granted the ADMINISTER KEY MANAGEMENT or SYSKM privilege.
6. Rekey the master encryption key of the relocated PDB.

   For example:

   ```
   ADMINISTER KEY MANAGEMENT SET KEY
   [FORCE KEYSTORE]
   IDENTIFIED BY EXTERNAL STORE|keystore_password
   WITH BACKUP [USING 'emp_key_backup'];
   ```

In this example, `FORCE KEYSTORE` is included because the keystore must be open during the rekey operation.

Before you rekey the master encryption key of the cloned PDB, the clone can still use master encryption keys that belong to the original PDB. However, these master encryption keys do not appear in the cloned PDB `V$` dynamic views. Rekeying the master encryption key ensures that the cloned PDB uses its own unique keys, which will be viewable in the `V$` views.

Related Topics

- Opening the TDE Wallet in a United Mode PDB
  To open a TDE wallet in united mode, you must use the `ADMINISTER KEY MANAGEMENT` statement with the `SET KEYSTORE OPEN` clause.
- *Oracle Multitenant Administrator's Guide*
- *Oracle Multitenant Administrator's Guide*
- *Oracle Database SQL Language Reference*

8.2.4 How Keystore Open and Close Operations Work in United Mode

You should be aware of how keystore open and close operations work in united mode.

For each PDB in united mode, you must explicitly open the password-protected TDE wallet or external keystore in the PDB to enable the Transparent Data Encryption operations to proceed. (Auto-login and local auto-login TDE wallets open automatically.) Closing a keystore on a PDB blocks all of the Transparent Data Encryption operations on that PDB.

The open and close keystore operations in a PDB depend on the open and close status of the keystore in the CDB root.

Note the following:

- You can create a separate keystore password for each PDB in united mode.
- Before you can manually open a password-protected TDE wallet or an external keystore in an individual PDB, you must open the wallet or keystore in the CDB root.
- If an auto-login TDE wallet is in use, or if the keystore is closed, then include the `FORCE KEYSTORE` clause in the `ADMINISTER KEY MANAGEMENT` statement when you open the wallet.
- If the keystore is a password-protected TDE wallet that uses an external store for passwords, then replace the password in the `IDENTIFIED BY` clause with `EXTERNAL STORE`.
- Before you can set a TDE master encryption key in an individual PDB, you must set the key in the CDB root. Oracle highly recommends that you include the `USING TAG` clause when you set keys in PDBs. For example:

```
SELECT ' ADMINISTER KEY MANAGEMENT SET KEY
USING TAG '''||SYS_CONTEXT('USERENV', 'CON_NAME')||' '||TO_CHAR (SYSDATE,
'YYYY-MM-DD HH24:MI:SS')||'''
FORCE KEYSTORE IDENTIFIED BY EXTERNAL STORE
WITH BACKUP CONTAINER = CURRENT;' AS "SET KEY COMMAND" FROM DUAL;
```

  Including the `USING TAG` clause enables you to quickly and easily identify the keys that belong to a certain PDB, and when they were created.
- Auto-login and local auto-login TDE wallets open automatically. You do not need to manually open these from the CDB root first, or from the PDB.

- If you close the keystore in the CDB root, then the keystores in the dependent PDBs also close. A keystore close operation in the root is the equivalent of performing a keystore close operation with the `CONTAINER` clause set to `ALL`.
- If you perform an `ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN` statement in the CDB root and set the `CONTAINER` clause to `ALL`, then the keystore will only be opened in each open PDB that is configured in united mode. Keystores for any PDBs that are configured in isolated mode are not opened.

8.2.5 Finding the Keystore Status for All of the PDBs in United Mode

You can create a convenience function that uses the `V$ENCRYPTION_WALLET` view to find the status for keystores in all PDBs in a CDB.

The `V$ENCRYPTION_WALLET` view displays the status of the keystore in a PDB, whether it is open, closed, uses a software or an external keystore, and so on.

- To create a function that uses the`V$ENCRYPTION_WALLET` view to find the keystore status, use the `CREATE PROCEDURE` PL/SQL statement.

Example 8-3 shows how to create this function.

Example 8-3 Function to Find the Keystore Status of All of the PDBs in a CDB

```
CREATE OR REPLACE PROCEDURE all_pdb_v$encryption_wallet
IS
    err_occ            BOOLEAN;
    curr_pdb           VARCHAR2(30);
    pdb_name           VARCHAR2(30);
    wrl_type           VARCHAR2(20);
    status             VARCHAR2(30);
    wallet_type        VARCHAR2(20);
    wallet_order       VARCHAR2(12);
    fully_backed_up    VARCHAR2(15);
    wrl_parameter      VARCHAR2(4000);
    cursor sel_pdbs IS SELECT NAME FROM V$CONTAINERS
                       WHERE NAME <> 'PDB$SEED' order by con_id desc;
  BEGIN

    -- Store the original PDB name
    SELECT sys_context('userenv', 'con_name') INTO curr_pdb FROM DUAL;
    IF curr_pdb <> 'CDB$ROOT' THEN
      dbms_output.put_line('Operation valid in ROOT only');
    END IF;

    err_occ := FALSE;
    dbms_output.put_line('---');
    dbms_output.put_line('PDB_NAME                       WRL_TYPE STATUS                         ');
    dbms_output.put_line('------------------------------ -------- ------------------------------');
    dbms_output.put_line('WALLET_TYPE          WALLET_ORDER FULLY_BACKED_UP');
    dbms_output.put_line('-------------------- ------------ ---------------');
    dbms_output.put_line('WRL_PARAMETER');
    dbms_output.put_line('--------------------------------------------------------------------------------');
    FOR pdbinfo IN sel_pdbs LOOP

      pdb_name := DBMS_ASSERT.ENQUOTE_NAME(pdbinfo.name, FALSE);
      EXECUTE IMMEDIATE 'ALTER SESSION SET CONTAINER = ' || pdb_name;

      BEGIN
        pdb_name := rpad(substr(pdb_name,1,30), 30, ' ');
        EXECUTE IMMEDIATE 'SELECT wrl_type from V$ENCRYPTION_WALLET' into wrl_type;
```

```
        wrl_type := rpad(substr(wrl_type,1,8), 8, ' ');
        EXECUTE IMMEDIATE 'SELECT status from V$ENCRYPTION_WALLET' into status;
        status := rpad(substr(status,1,30), 30, ' ');
        EXECUTE IMMEDIATE 'SELECT wallet_type from V$ENCRYPTION_WALLET' into wallet_type;
        wallet_type := rpad(substr(wallet_type,1,20), 20, ' ');
        EXECUTE IMMEDIATE 'SELECT wallet_order from V$ENCRYPTION_WALLET' into wallet_order;
        wallet_order := rpad(substr(wallet_order,1,9), 12, ' ');
        EXECUTE IMMEDIATE 'SELECT fully_backed_up from V$ENCRYPTION_WALLET' into fully_backed_up;
        fully_backed_up := rpad(substr(fully_backed_up,1,9), 15, ' ');
        EXECUTE IMMEDIATE 'SELECT wrl_parameter from V$ENCRYPTION_WALLET' into wrl_parameter;
        wrl_parameter := rpad(substr(wrl_parameter,1,79), 79, ' ');
        dbms_output.put_line(pdb_name || ' ' || wrl_type || ' ' || status);
        dbms_output.put_line(wallet_type || ' ' || wallet_order || ' ' || fully_backed_up);
        dbms_output.put_line(wrl_parameter);

      EXCEPTION
        WHEN OTHERS THEN
        err_occ := TRUE;
      END;
    END LOOP;

    IF err_occ = TRUE THEN
       dbms_output.put_line('One or more PDB resulted in an error');
    END IF;
  END;
.
/
set serveroutput on
exec all_pdb_v$encryption_wallet;
```

9
Administering Isolated Mode

Administering isolated mode means managing the keystores, master encryption keys, and general Transparent Database Encryption (TDE) functionality.

9.1 Administering Keystores and TDE Master Encryption Keys in Isolated Mode

After you create a keystore and a TDE master encryption key in isolated mode, you can perform administration tasks such as rekeying or tagging encryption keys.

To change the password of an external keystore, you must use the administrative interface of the external keystore. You cannot perform this operation by using the `ADMINISTER KEY MANAGEMENT` statement.

9.1.1 Changing the Keystore Password in Isolated Mode

You can change the password of a TDE wallet when the PDB is in isolated mode.

To change the password of an external keystore, you must use the administrative interface of the external keystore. You cannot perform this operation by using the `ADMINISTER KEY MANAGEMENT` statement.

9.1.1.1 Changing the Password-Protected TDE Wallet Password in Isolated Mode

To change the password of a password-protected TDE wallet in isolated mode, you must use the `ADMINISTER KEY MANAGEMENT` statement.

You can change this password at any time, as per the security policies, compliance guidelines, and other security requirements of your site. As part of the command to change the password, you will be forced to specify the `WITH BACKUP` clause, and thus forced to make a backup of the current TDE wallet. During the password change operation, Transparent Data Encryption operations such as encryption and decryption will continue to work normally. You can change this password at any time. You may want to change this password if you think it was compromised.

1. Connect to the isolated mode PDB as a user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.
2. Change the password for the TDE wallet by using the following syntax:

```
ADMINISTER KEY MANAGEMENT ALTER KEYSTORE PASSWORD
[FORCE KEYSTORE]
IDENTIFIED BY
old_password SET new_password
[WITH BACKUP [USING 'backup_identifier']];
```

In this specification:

- `FORCE KEYSTORE` temporarily opens the password-protected TDE wallet for this operation if the TDE wallet is closed, if an auto-login TDE wallet is configured and is currently open, or if a password-protected TDE wallet is configured and is currently closed.
- *`old_password`* is the current TDE wallet password that you want to change.
- *`new_password`* is the new password that you set for the TDE wallet.

The following example creates a backup of the TDE wallet and then changes the TDE wallet password:

```
ADMINISTER KEY MANAGEMENT ALTER KEYSTORE PASSWORD
IDENTIFIED BY
old_password SET new_password
WITH BACKUP USING 'pwd_change';

keystore altered.
```

This example performs the same operation but uses the `FORCE KEYSTORE` clause in case the auto-login TDE wallet is in use or the password-protected TDE wallet is closed.

```
ADMINISTER KEY MANAGEMENT ALTER KEYSTORE PASSWORD
FORCE KEYSTORE
IDENTIFIED BY
old_password SET new_password
WITH BACKUP USING 'pwd_change';

keystore altered.
```

Related Topics

- Performing Operations That Require a Keystore Password
  Many `ADMINISTER KEY MANAGEMENT` operations require access to a keystore password, for both TDE wallets and external keystores.

9.1.1.2 Changing the Password of an External Keystore in Isolated Mode

To change the password of an external keystore, you must close the external keystore and then change the password from the external keystore's management interface.

1. Connect to the isolated mode PDB as a user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.
2. Close the external keystore.
   - For example:

     ```
     ADMINISTER KEY MANAGEMENT SET KEYSTORE CLOSE
     IDENTIFIED BY "external_key_manager_password";
     ```

   - For an external keystore whose password is stored externally:

     ```
     ADMINISTER KEY MANAGEMENT SET KEYSTORE CLOSE
     IDENTIFIED BY EXTERNAL STORE;
     ```

3. Update the credentials of the external store to use "*`new_external_key_manager_password`*".

   Currently, the external store contains the old credentials, which would no longer work.

   For example:

   ```
   ADMINISTER KEY MANAGEMENT
   UPDATE SECRET 'new_external_key_manager_password'
   ```

```
FOR CLIENT 'TDE_WALLET'
TO LOCAL AUTO_LOGIN KEYSTORE '/etc/ORACLE/WALLETS/orcl/external_store';
```

4. Open the external keystore.

   For example:

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

   For an external keystore whose password is stored externally:

   ```
   ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN
   IDENTIFIED BY EXTERNAL STORE;
   ```

Related Topics

- Performing Operations That Require a Keystore Password
  Many `ADMINISTER KEY MANAGEMENT` operations require access to a keystore password, for both TDE wallets and external keystores.

9.1.2 Backing Up a Password-Protected TDE Wallet in Isolated Mode

The `BACKUP KEYSTORE` clause of the `ADMINISTER KEY MANAGEMENT` statement backs up a password-protected TDE wallet.

1. Connect to the isolated mode PDB as a user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.
2. Back up the keystore by using the following syntax:

   ```
   ADMINISTER KEY MANAGEMENT BACKUP KEYSTORE
   [USING 'backup_identifier']
   [FORCE KEYSTORE]
   IDENTIFIED BY [EXTERNAL STORE | TDE_wallet_password]
   [TO 'TDE_wallet_location'];
   ```

   In this specification:

   - `USING` *`backup_identifier`* is an optional string that you can provide to identify the backup. Enclose this identifier in single quotation marks (`' '`). This identifier is appended to the named keystore file (for example, `ewallet_`*`time-stamp`*`_emp_key_backup.p12`).
   - `FORCE KEYSTORE` temporarily opens the password-protected TDE wallet for this operation. You must open the TDE wallet for this operation.
   - `IDENTIFIED BY` is required for the `BACKUP KEYSTORE` operation on a password-protected keystore because although the backup is simply a copy of the existing keystore, the status of the TDE master encryption key in the password-protected keystore must be set to `BACKED UP` and for this change the keystore password is required.
   - *`TDE_wallet_location`* is the path at which the backup TDE wallet is stored. This setting is restricted to the PDB when the PDB lockdown profile `EXTERNAL_FILE_ACCESS` setting is blocked in the PDB or when the `PATH_PREFIX` variable was set when the PDB was created. If you do not specify the *`TDE_wallet_location`*, then the backup is created in the same directory as the original TDE wallet. Enclose this location in single quotation marks (`' '`).

   The following example backs up a TDE wallet in the same location as the source TDE wallet.

```
ADMINISTER KEY MANAGEMENT BACKUP KEYSTORE
USING 'hr.emp_keystore'
FORCE KEYSTORE
IDENTIFIED BY
'TDE_wallet_password';

keystore altered.
```

In the following version, the password for the TDE wallet is stored externally, so the `EXTERNAL STORE` clause is used.

```
ADMINISTER KEY MANAGEMENT BACKUP KEYSTORE
USING 'hr.emp_keystore'
FORCE KEYSTORE
IDENTIFIED BY EXTERNAL STORE;
```

After you run this statement, an `ewallet_identifier.p12` file (for example, `ewallet_time-stamp_hr.emp_keystore.p12`) appears in the TDE wallet backup location.

Related Topics

- Backing Up Password-Protected TDE Wallets
  When you back up a password-protected TDE wallet, you can create a backup identifier string to describe the backup type.

9.1.3 Merging TDE Wallets in Isolated Mode

In isolated mode, you can merge TDE wallets.

9.1.3.1 Merging One TDE Wallet into an Existing TDE Wallet in Isolated Mode

In isolated mode, you can use the `ADMINISTER KEY MANAGEMENT` statement with the `MERGE KEYSTORE` clause to merge one TDE wallet into another existing TDE wallet.

1. Connect to the isolated mode PDB as a user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.
2. Merge the TDE wallets by using the following syntax:

```
ADMINISTER KEY MANAGEMENT MERGE KEYSTORE 'TDE_wallet1_location'
[IDENTIFIED BY TDE_wallet1_password]
INTO EXISTING KEYSTORE 'TDE_wallet2_location'
IDENTIFIED BY TDe_wallet2_password
[WITH BACKUP [USING 'backup_identifier]];
```

In this specification:

- *TDE_wallet1_location* is the directory location of the first TDE wallet, which will be left unchanged after the merge. Enclose this path in single quotation marks (`' '`).
- The `IDENTIFIED BY` clause is required for the first TDE wallet if it is a password-protected TDE wallet. *TDE_wallet1_password* is the password for the first TDE wallet.
- *TDE_wallet2_location* is the directory location of the second TDE wallet into which the first TDE wallet is to be merged. Enclose this path in single quotation marks (`' '`).
- *TDE_wallet2_password* is the password for the second TDE wallet.

The target TDE wallet (*TDE_wallet2*) remains a password-protected TDE wallet after the TDE wallet merge operation.

Related Topics

- About Merging TDE Wallets
  You can merge any combination of TDE wallets, but the merged keystore must be password-protected. It can have a password that is different from the constituent wallets.

9.1.3.2 Merging Two TDE Wallets into a Third New TDE Wallet in Isolated Mode

In isolated mode, you can merge two TDE wallets into a third new TDE wallet, so that the two existing TDE wallets are not changed and the new TDE wallet contains the keys of both source TDE wallets.

1. Connect to the isolated mode PDB as a user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.
2. Merge the TDE wallets by using the following syntax:

```
ADMINISTER KEY MANAGEMENT MERGE KEYSTORE 'TDE_wallet1_location'
[IDENTIFIED BY TDE_wallet1_password] AND TDE_wallet 'TDE_wallet2_location'
[IDENTIFIED BY TDE_wallet2_password]
INTO NEW KEYSTORE 'TDE_wallet3_location'
IDENTIFIED BY TDE_wallet3_password;
```

   In this specification:

   - *TDE_wallet1_location* is the directory location of the first TDE wallet, which will be left unchanged after the merge. Enclose this path in single quotation marks (`' '`).
   - The `IDENTIFIED BY` clause is required for the first TDE wallet if it is a password-protected TDE wallet. *TDE_wallet1_password* is the current password for the first TDE wallet.
   - *TDE_wallet2_location* is the directory location of the second TDE wallet. Enclose this path in single quotation marks (`' '`).
   - The `IDENTIFIED BY` clause is required for the second TDE wallet if it is a password-protected TDE wallet. *TDE_wallet2_password* is the current password for the second TDE wallet.
   - *TDE_wallet3_location* specifies the directory location of the new, merged TDE wallet. Enclose this path in single quotation marks (`' '`). If there is already an existing TDE wallet at this location, the command exits with an error.
   - *TDE_wallet3_password* is the new password for the merged TDE wallet.

   The following example merges an auto-login TDE wallet with a password-protected TDE wallet to create a merged password-protected TDE wallet at a new location:

```
ADMINISTER KEY MANAGEMENT MERGE KEYSTORE '/etc/ORACLE/KEYSTORE/DB1'
AND KEYSTORE '/etc/ORACLE/KEYSTORE/DB2'
IDENTIFIED BY existing_password_for_TDE_wallet_2
INTO NEW KEYSTORE '/etc/ORACLE/KEYSTORE/DB3'
IDENTIFIED BY new_password_for_TDE_wallet_3;

keystore altered.
```

Related Topics

- About Merging TDE Wallets
  You can merge any combination of TDE wallets, but the merged keystore must be password-protected. It can have a password that is different from the constituent wallets.

9.1.4 Closing Keystores in Isolated Mode

You can close both software and external keystores in isolated mode, unless the system tablespace is encrypted.

9.1.4.1 Closing a TDE Wallet in Isolated Mode

You can close password-protected TDE wallets, auto-login TDE wallets, and local auto-login TDE wallets in isolated mode.

In the case of an auto-login TDE wallet, which opens automatically when it is accessed, you must first move it to a new location where it cannot be automatically opened, then you must manually close it. You must do this if you are changing your configuration from an auto-login TDE wallet to a password-protected TDE wallet: you change the configuration to stop using the auto-login TDE wallet (by moving the auto-login TDE wallet to another location where it cannot be automatically opened), and then closing the auto-login TDE wallet.

1. Connect to the isolated mode PDB as a user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.
2. Close the TDE wallet by using the following syntax.

   Note that the only difference between the following two `ADMINISTER KEY MANAGEMENT SET KEYSTORE CLOSE` statements is that a password must be provided for a password-protected TDE wallet.

   - For a password-protected TDE wallet:

     ```
     ADMINISTER KEY MANAGEMENT SET KEYSTORE CLOSE
     IDENTIFIED BY [EXTERNAL STORE | TDE_wallet_password];
     ```

     Closing a password-protected TDE wallet disables all encryption and decryption operations. Any attempt to encrypt or decrypt data or access encrypted data results in an error.

   - For an auto-login or local auto-login TDE wallet:

     ```
     ADMINISTER KEY MANAGEMENT SET KEYSTORE CLOSE;
     ```

     The result of this statement will not necessarily be that the TDE wallet status will change to `CLOSED`, because unless you also moved the `cwallet.sso` file to a location that Oracle Database cannot find, then a background job or background process could automatically re-open the auto-login TDE wallet. This can cause the status to potentially always appear to be `OPEN` even after the `ADMINISTER KEY MANAGEMENT SET KEYSTORE CLOSE` statement completed successfully.

Related Topics

- About Closing Keystores
  After you open a keystore, it remains open until you shut down the database instance.

9.1.4.2 Closing an External Keystore in Isolated Mode

To close an external keystore, you must use the `ADMINISTER KEY MANAGEMENT` statement with the `SET KEYSTORE CLOSE` clause.

For an Oracle Key Vault keystore, you can only provide the password. No user name is allowed in the `IDENTIFIED BY` clause. Enclose the password in double quotation marks.

1. Connect to the isolated mode PDB as a user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.
2. Close the external keystore by using the following syntax:

```
ADMINISTER KEY MANAGEMENT SET KEYSTORE CLOSE
IDENTIFIED BY [EXTERNAL STORE | "external_key_manager_password"];
```

Closing a keystore disables all encryption and decryption operations. Any attempt to encrypt or decrypt data or access encrypted data results in an error.

Related Topics

- About Closing Keystores
  After you open a keystore, it remains open until you shut down the database instance.

9.1.5 Creating a User-Defined TDE Master Encryption Key in Isolated Mode

To create a user-defined TDE master encryption key, use the `ADMINISTER KEY MANAGEMENT` statement with the `SET | CREATE [ENCRYPTION] KEY` clause.

1. Connect to the isolated mode PDB as a user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.
2. Create the user-defined TDE master encryption key by using the following syntax:

```
ADMINISTER KEY MANAGEMENT SET | CREATE [ENCRYPTION] KEY
'mkid:mk | mk'
[USING ALGORITHM 'algorithm']
[FORCE KEYSTORE]
IDENTIFIED BY [EXTERNAL STORE | keystore_password]
[WITH BACKUP [USING 'backup_identifier']];
```

In this specification:

- `SET | CREATE` : Enter `SET` if you want to create the master and activate the TDE master encryption key now, or enter `CREATE` if you want to create the key for later use, without activating it yet.
- *mkid* and *mk*:
  - *mkid*, the TDE master encryption key ID, is a 16–byte hex-encoded value that you can specify or have Oracle Database generate.
  - *mk*, the TDE master encryption key, is a hex-encoded value that you can specify or have Oracle Database generate, either 32 bytes (for the for `AES256`, `ARIA256`, and `GOST256` algorithms) or 16 bytes (for the `SEED128` algorithm).

> **Note:**
>
> Starting with Oracle Database 23ai, the Transparent Data Encryption (TDE) decryption libraries for the GOST and SEED algorithms are deprecated, and encryption to GOST and SEED are desupported. Starting with Oracle Database 23ai, the Transparent Data Encryption (TDE) encryption libraries for the GOST and SEED algorithms are desupported and removed. The GOST and SEED decryption libraries are deprecated. Both are removed on HP Itanium platforms.
> GOST 28147-89 has been deprecated by the Russian government, and SEED has been deprecated by the South Korean government. If you need South Korean government-approved TDE cryptography, then use ARIA instead. If you are using GOST 28147-89, then you must decrypt and encrypt with another supported TDE algorithm. The decryption algorithms for GOST 28147-89 and SEED are included with Oracle Database 23ai, but are deprecated, and the GOST encryption algorithm is desupported with Oracle Database 23ai. If you are using GOST or SEED for TDE encryption, then Oracle recommends that you perform an online rekey operation before upgrading to Oracle Database 23ai. However, with the exception of the HP Itanium platform, the GOST and SEED decryption libraries are available with Oracle Database 23ai, so you can also decrypt after upgrading.

If you omit the *mkid*:*mk*|*mkid* clause but include the *mk* value, then Oracle Database generates the *mkid* for the *mk*.

If you omit the entire *mkid*:*mk*|*mkid* clause, then Oracle Database generates these values for you.

- USING ALGORITHM: Specify one of the following supported algorithms:
  - AES256
  - ARIA256
  - SEED128 (deprecated)
  - GOST256 (deprecated)

  If you omit the algorithm, then the default, AES256, is used.
- FORCE KEYSTORE temporarily opens the password-protected TDE wallet for this operation. You must open the TDE wallet for this operation.

The following example includes a user-created TDE master encryption key but no TDE master encryption key ID, so that the TDE master encryption key ID is generated:

```
ADMINISTER KEY MANAGEMENT SET ENCRYPTION KEY
'3D432109DF88967A541967062A6F4E460E892318E307F017BA048707B402493C'
USING ALGORITHM 'ARIA256'
FORCE KEYSTORE
IDENTIFIED BY keystore_password WITH BACKUP;
```

The next example creates user-defined keys for both the master encryption ID and the TDE master encryption key. It omits the algorithm specification, so the default algorithm AES256 is used.

```
ADMINISTER KEY MANAGEMENT CREATE ENCRYPTION KEY
'10203040506070801112131415161718:3D432109DF88967A541967062A6F4E460E892318E307F017BA0
```

```
48707B402493C'
IDENTIFIED BY keystore_password WITH BACKUP;
```

Related Topics

- About User-Defined TDE Master Encryption Keys
  A TDE master encryption key that is outside the database has its own user-generated ID, which tracks the use of the TDE master encryption key.
- Supported Encryption and Integrity Algorithms
  Oracle supports the AES, ARIA and DES algorithms.

9.1.6 Creating a TDE Master Encryption Key for Later Use in Isolated Mode

A keystore must be open before you can create a TDE master encryption key for use later on in isolated mode.

1. Connect to the isolated mode PDB as a user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.
2. Create the TDE master encryption key by using the following syntax:

   ```
   ADMINISTER KEY MANAGEMENT CREATE KEY [USING TAG 'tag']
   [FORCE KEYSTORE]
   IDENTIFIED BY EXTERNAL STORE | keystore_password
   WITH BACKUP [USING 'backup_identifier'];
   ```

   In this specification:

   - `FORCE KEYSTORE` temporarily opens the password-protected TDE wallet for this operation. You must open the TDE wallet for this operation.
   - `IDENTIFIED BY` is required for the `BACKUP KEYSTORE` operation on a password-protected keystore because although the backup is simply a copy of the existing keystore, the status of the TDE master encryption key in the password-protected keystore must be set to `BACKED UP` and for this change the keystore password is required.
   - *keystore_location* is the path at which the backup keystore is stored. This setting is restricted to the PDB when the PDB lockdown profile `EXTERNAL_FILE_ACCESS` setting is blocked in the PDB or when the `PATH_PREFIX` variable was not set when the PDB was created. If you do not specify the *keystore_location*, then the backup is created in the same directory as the original keystore. Enclose this location in single quotation marks (`' '`).

   For example:

   ```
   ADMINISTER KEY MANAGEMENT CREATE KEY
   FORCE KEYSTORE
   IDENTIFIED BY keystore_password
   WITH BACKUP;
   ```

3. If necessary, activate the TDE master encryption key.
   a. Find the key ID.

      ```
      SELECT KEY_ID FROM V$ENCRYPTION_KEYS;

      KEY_ID
      ------------------------------------------------
      AWsHwVYC2U+Nv3RVphn/yAIAAAAAAAAAAAAAAAAAAAAAAAAAAAA
      ```

   b. Use this key ID to activate the key.

```
ADMINISTER KEY MANAGEMENT USE KEY
'AWsHwVYC2U+Nv3RVphn/yAIAAAAAAAAAAAAAAAAAAAAAAAAAAAAA'
USING TAG 'quarter:second;description:Activate Key on standby'
IDENTIFIED BY password
WITH BACKUP;
```

Related Topics

- About Creating a TDE Master Encryption Key for Later Use
  The `CREATE KEY` clause of the `ADMINISTER KEY MANAGEMENT` statement can create a TDE master encryption key to be activated at a later date.

9.1.7 Activating a TDE Master Encryption Key in Isolated Mode

To activate a TDE master encryption key in isolated mode, you must open the keystore and use `ADMINISTER KEY MANAGEMENT` with the `USE KEY` clause.

1. Connect to the isolated mode PDB as a user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.
2. Query the `KEY_ID` column of the `V$ENCRYPTION_KEYS` view to find the key identifier.

   For example:

   ```
   SELECT KEY_ID FROM V$ENCRYPTION_KEYS;

   KEY_ID
   ------------------------------------------------
   ARaHD762tUkkvyLgPzAi6hMAAAAAAAAAAAAAAAAAAAAAAAAAAAA
   ```

3. Use this key identifier to activate the TDE master encryption key by using the following syntax:

   ```
   ADMINISTER KEY MANAGEMENT USE KEY 'key_identifier_from_V$ENCRYPTION_KEYS'
   [USING TAG 'tag']
   [FORCE KEYSTORE]
   IDENTIFIED BY [EXTERNAL STORE | keystore_password]
   [WITH BACKUP [USING 'backup_identifier']];
   ```

   In this specification:

   - `FORCE KEYSTORE` temporarily opens the password-protected TDE wallet for this operation. You must open the TDE wallet for this operation.

   For example:

   ```
   ADMINISTER KEY MANAGEMENT USE KEY
   'ARaHD762tUkkvyLgPzAi6hMAAAAAAAAAAAAAAAAAAAAAAAAAAAAA'
   FORCE KEYSTORE
   IDENTIFIED BY keystore_password
   WITH BACKUP;
   ```

Related Topics

- About Activating TDE Master Encryption Keys
  You can activate a previously created or imported TDE master encryption key by using the `USE KEY` clause of `ADMINISTER KEY MANAGEMENT`.

9.1.8 Rekeying the TDE Master Encryption Key in Isolated Mode

You can use the `ADMINISTER KEY MANAGEMENT` statement with the `SET KEY` clause to rekey a TDE master encryption key.

1. Connect to the isolated mode PDB as a user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.
2. If you are rekeying the TDE master encryption key for a keystore that has auto login enabled, then ensure that both the auto login keystore, identified by the `.sso` file, and the encryption keystore, identified by the `.p12` file, are present.

   You can find the location of these files by querying the `WRL_PARAMETER` column of the `V$ENCRYPTION_WALLET` view. To find the `WRL_PARAMETER` values for all of the database instances, query the `GV$ENCRYPTION_WALLET` view.
3. Rekey the TDE master encryption key by using the following syntax:

```
ADMINISTER KEY MANAGEMENT SET [ENCRYPTION] KEY
[FORCE KEYSTORE]
[USING TAG 'tag_name']
IDENTIFIED BY [EXTERNAL STORE | keystore_password]
[WITH BACKUP [USING 'backup_identifier']];
```

   In this specification:

   - *tag* is the associated attributes and information that you define. Enclose this setting in single quotation marks (`' '`).
   - `FORCE KEYSTORE` temporarily opens the password-protected TDE wallet for this operation. You must open the TDE wallet for this operation.
   - `IDENTIFIED BY` can be one of the following settings:
     - `EXTERNAL STORE` uses the keystore password stored in the external store to perform the keystore operation.
     - *keystore_password* is the password that was created for this keystore.

   For example:

```
ADMINISTER KEY MANAGEMENT SET KEY
FORCE KEYSTORE
IDENTIFIED BY keystore_password
WITH BACKUP USING 'emp_key_backup';

keystore altered.
```

Related Topics

- About Rekeying the TDE Master Encryption Key
  Oracle Database uses a unified TDE Master Encryption Key for both TDE column encryption and TDE tablespace encryption.

9.1.9 Moving a TDE Master Encryption Key into a New Keystore in Isolated Mode

In isolated mode, you can move an existing TDE master encryption key into a new TDE wallet from an existing password TDE wallet.

This feature enables you to move a subset, or all, of a TDE wallet's keys into a new TDE wallet. After you move the key or keys to the new TDE wallet and then back up the old TDE wallet, optionally you then can delete this old TDE wallet.

1. Connect to the isolated mode PDB as a user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.

2. Query the `CREATION_TIME` and `KEY_ID` columns of the `V$ENCRYPTION_KEYS` view to find the key identifier of the key that you want to move.

   For example:

```
SELECT CREATION_TIME, KEY_ID FROM V$ENCRYPTION_KEYS;

CREATION TIME
---------------------------------------------------
22-SEP-19 08.55.12.956170 PM +00:00

KEY_ID
---------------------------------------------------
ARaHD762tUkkvyLgPzAi6hMAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```

3. Move the key into a new TDE wallet by using the following syntax:

```
ADMINISTER KEY MANAGEMENT
MOVE [ENCRYPTION] KEYS
TO NEW KEYSTORE 'TDE_wallet_location1'
IDENTIFIED BY TDE_wallet1_password
FROM [FORCE] KEYSTORE
IDENTIFIED BY TDE_wallet2_password
[WITH IDENTIFIER IN
{ 'key_identifier' [, 'key_identifier' ]... | ( subquery ) }]
[WITH BACKUP [USING 'backup_identifier'] ];
```

   In this specification:

   - *TDE_wallet_location1* is the path to the `wallet` directory that will store the new TDE wallet `.p12` file. By default, this directory is in `$ORACLE_BASE/admin/`*db_unique_name*`/wallet`.
   - *TDE_wallet1_password* is the password for the TDE wallet from which the new TDE wallet is moved.
   - `FORCE` temporarily opens the TDE wallet for this operation.
   - *TDE_wallet2_password* is the password for the TDE wallet from which the key is moving.
   - *subquery* can be used to find the exact key identifier that you want.
   - *backup_identifier* is an optional description of the backup. Enclose *backup_identifier* in single quotation marks (`' '`).

   For example:

```
ADMINISTER KEY MANAGEMENT MOVE KEYS
TO NEW KEYSTORE '$ORACLE_BASE/admin/orcl/wallet'
IDENTIFIED BY TDE_wallet1_password
FROM FORCE KEYSTORE
IDENTIFIED BY TDE_wallet2_password
WITH IDENTIFIER IN
(SELECT KEY_ID FROM V$ENCRYPTION_KEYS WHERE ROWNUM < 2)
WITH BACKUP;
```

   After the keys are moved to the new TDE wallet, they no longer exist in the old TDE wallet.

4. To delete the old TDE wallet, go to the `wallet` directory and do the following:

   a. Back up the `.p12` file containing the TDE wallet that you want to delete.

   b. Manually delete the `.p12` file containing the TDE wallet.

To find the location of the TDE wallet, open the TDE wallet, and then query the `WRL_PARAMETER` column of the `V$ENCRYPTION_WALLET` view.

Related Topics

- About Moving TDE Master Encryption Keys into a New Keystore
  You can move an unused (and safely archived) TDE master encryption key into a new keystore.
- Dangers of Deleting TDE Wallets
  Oracle strongly recommends that you do not delete TDE wallets.

9.1.10 Creating a Custom Attribute Tag in Isolated Mode

To create a custom attribute tag in isolated mode, you must use the `SET TAG` clause of the `ADMINISTER KEY MANAGEMENT` statement.

1. Connect to the isolated mode PDB as a user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.
2. If necessary, query the `TAG` column of the `V$ENCRYPTION_KEY` dynamic view to find a listing of existing tags for the TDE master encryption keys.

   When you create a new tag for a TDE master encryption key, it overwrites the existing tag for that TDE master encryption key.
3. Create the custom attribute tag by using the following syntax:

```
ADMINISTER KEY MANAGEMENT SET TAG 'tag'
FOR 'master_key_identifier'
[FORCE KEYSTORE]
IDENTIFIED BY [EXTERNAL STORE | keystore_password]
[WITH BACKUP [USING 'backup_identifier']];
```

   In this specification:

   - *tag* is the associated attributes or information that you define. Enclose this information in single quotation marks (`' '`).
   - *master_key_identifier* identifies the TDE master encryption key for which the *tag* is set. To find a list of TDE master encryption key identifiers, query the `KEY_ID` column of the `V$ENCRYPTION_KEYS` dynamic view.
   - `FORCE KEYSTORE` temporarily opens the password-protected TDE wallet for this operation. You must open the TDE wallet for this operation.
   - `IDENTIFIED BY` can be one of the following settings:
     - `EXTERNAL STORE` uses the keystore password stored in the external store to perform the keystore operation.
     - *keystore_password* is the password that was created for this keystore.
     - *backup_identifier* defines the tag values. Enclose this setting in single quotation marks (`' '`) and separate each value with a colon.

   For example, to create a tag that uses two values, one to capture a specific session ID and the second to capture a specific terminal ID:

```
ADMINISTER KEY MANAGEMENT SET ENCRYPTION KEY
USING TAG 'sessionid=3205062574:terminal=xcvt'
IDENTIFIED BY keystore_password
WITH BACKUP;
```

```
keystore altered.
```

Both the session ID (`3205062574`) and terminal ID (`xcvt`) can derive their values by using either the `SYS_CONTEXT` function with the `USERENV` namespace, or by using the `USERENV` function.

Related Topics

- About Creating Custom Attribute Tags
  Attribute tags enable you to monitor specific activities users perform, such as accessing a particular terminal ID.

9.1.11 Exporting and Importing the TDE Master Encryption Key in Isolated Mode

You can export and import the TDE master encryption key in different ways in isolated mode.

9.1.11.1 Exporting a TDE Master Encryption Key in Isolated Mode

In isolated mode, you can use the `ADMINISTER KEY MANAGEMENT` statement to export a TDE master encryption key.

1. Connect to the isolated mode PDB as a user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.
2. Export the TDE master encryption keystore by using the following syntax:

```
ADMINISTER KEY MANAGEMENT EXPORT [ENCRYPTION] KEYS
WITH SECRET "export_secret"
TO 'file_path'
[FORCE KEYSTORE]
IDENTIFIED BY [EXTERNAL STORE | keystore_password]
[WITH IDENTIFIER IN 'key_id1', 'key_id2', 'key_idn' | (SQL_query)];
```

In this specification:

- *export_secret* is a password that you can specify to encrypt the export the file that contains the exported keys. Enclose this secret in double quotation marks (`" "`), or you can omit the quotation marks if the secret has no spaces.
- *file_path* is the complete path and name of the file to which the keys must be exported. Enclose this path in single quotation marks (`' '`). You can export to regular file systems only.
- `FORCE KEYSTORE` temporarily opens the password-protected TDE wallet for this operation. You must open the TDE wallet for this operation.
- *key_id1*, *key_id2*, *key_idn* is a string of one or more TDE master encryption key identifiers for the TDE master encryption key being exported. Separate each key identifier with a comma and enclose each of these key identifiers in single quotation marks (`' '`). To find TDE master encryption key identifiers, query the `KEY_ID` column of the `V$ENCRYPTION_KEYS` dynamic view.
- *SQL_query* is a query that fetches a list of the TDE master encryption key identifiers. It should return only one column that contains the TDE master encryption key identifiers. This query is run with current user rights.

Related Topics

- Exporting and Importing the TDE Master Encryption Key
  You can export and import the TDE master encryption key in different ways.

9.1.11.2 Importing a TDE Master Encryption Key in Isolated Mode

The `ADMINISTER KEY MANAGEMENT` statement with the `IMPORT [ENCRYPTION] KEYS WITH SECRET` clause can import a TDE master encryption key.

1. Connect to the isolated mode PDB as a user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.
2. Import the TDE master encryption keystore by using the following syntax:

```
ADMINISTER KEY MANAGEMENT IMPORT [ENCRYPTION] KEYS
WITH SECRET "import_secret"
FROM 'file_name'
[FORCE KEYSTORE]
IDENTIFIED BY [EXTERNAL STORE | keystore_password]
[WITH BACKUP [USING 'backup_identifier']];
```

   In this specification:

   - *import_secret* is the same password that was used to encrypt the keys during the export operation. Enclose this secret in double quotation marks (`" "`), or you can omit the quotation marks if the secret has no spaces.
   - *file_name* is the complete path and name of the file from which the keys need to be imported. Enclose this setting in single quotation marks (`' '`).
   - `FORCE KEYSTORE` temporarily opens the password-protected TDE wallet for this operation. You must open the TDE wallet for this operation.

Related Topics

- Exporting and Importing the TDE Master Encryption Key
  You can export and import the TDE master encryption key in different ways.

9.1.12 Storing Oracle Database Secrets in Isolated Mode

Secrets are data that support internal Oracle Database features that integrate external clients such as Oracle GoldenGate into the database.

9.1.12.1 About Storing Oracle Database Secrets in a Keystore in Isolated Mode

Keystores (both TDE wallets and external keystores) can store secrets that support internal Oracle Database features and integrate external clients such as Oracle GoldenGate.

The secret key must be a string adhering to Oracle identifier rules. You can add, update, or delete a client secret in an existing keystore. The Oracle GoldenGate Extract process must have data encryption keys to decrypt the data that is in data files and in `REDO` or `UNDO` logs. Keys are encrypted with shared secrets when you share the keys between an Oracle database and an Oracle GoldenGate client. The TDE wallet stores the shared secrets.

Depending on your site's requirements, you may require automated open keystore operations even when an external keystore is configured. For this reason, the external key manager password can be stored in an auto-login TDE wallet, which enables the auto-login capability for the external key manager. The Oracle Database side can also store the credentials for the database to log in to an external storage server in the TDE wallet.

You can store Oracle Database secrets in both TDE wallets and external keystores:

- **TDE wallets:** You can store secrets in password-based, auto-login, and local auto-login TDE wallets. If you want to store secrets in an auto-login (or auto-login local) TDE wallet, then note the following:
  - If the auto-login TDE wallet is in the same location as its corresponding password-based TDE wallet, then the secrets are added automatically.
  - If the auto-login TDE wallet is in a different location from its corresponding password-based TDE wallet, then you must create the auto-login TDE wallet again from the password-based TDE walllet, and keep the two walltets in synchronization.
- **External keystores:** You can store secrets in standard external key managers.

Related Topics

- Configuring Auto-Open Connections into External Key Managers
  An external key manager can be configured to use the auto-login capability.

9.1.12.2 Storing Oracle Database Secrets in a TDE Wallet in Isolated Mode

The `ADMINISTER KEY MANAGEMENT ADD SECRET|UPDATE SECRET|DELETE SECRET` statements can add secrets, update secrets, and delete secrets in a TDE wallet.

1. Connect to the isolated mode PDB as a user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.
2. Add, update, or delete a database secret in a TDE wallet by using the following syntax:
   - To add a secret:

```
ADMINISTER KEY MANAGEMENT
ADD SECRET 'secret' FOR CLIENT 'client_identifier'
[USING TAG 'tag']
[TO [[LOCAL] AUTO_LOGIN] KEYSTORE TDE_wallet_location
[WITH BACKUP [USING backup_id]]
| [FORCE KEYSTORE]
IDENTIFIED BY [EXTERNAL STORE | TDE_wallet_password]
[WITH BACKUP [USING backup_id]]];
```

   - To update a secret:

```
ADMINISTER KEY MANAGEMENT
UPDATE SECRET 'secret' FOR CLIENT 'client_identifier'
[USING TAG 'tag']
[TO [[LOCAL] AUTO_LOGIN] KEYSTORE TDE_wallet_location
[WITH BACKUP [USING backup_id]]
| [FORCE KEYSTORE]
IDENTIFIED BY [EXTERNAL STORE | TDE_wallet_password]
[WITH BACKUP [USING backup_id]]];
```

   - To delete a secret:

```
ADMINISTER KEY MANAGEMENT
DELETE SECRET FOR CLIENT 'client_identifier'
[FROM [[LOCAL] AUTO_LOGIN] KEYSTORE TDE_wallet_location
[FORCE KEYSTORE]
IDENTIFIED BY [EXTERNAL STORE | TDE_wallet_password]
[WITH BACKUP [USING backup_id]]];
```

   The specification is as follows:

- *secret* is the client secret key to be stored, updated, or deleted. To find information about existing secrets and their client identifiers, query the V$CLIENT_SECRETS dynamic view.
- *client_identifier* is an alphanumeric string used to identify the secret key.
- FORCE KEYSTORE temporarily opens the password-protected TDE wallet for this operation. You must open the TDE wallet for this operation.

Related Topics

- Storing Oracle Database Secrets in Isolated Mode
  Secrets are data that support internal Oracle Database features that integrate external clients such as Oracle GoldenGate into the database.

9.1.12.3 Example: Adding an Oracle Key Vault Password to a TDE Wallet

The ADMINISTER KEY MANAGEMENT ADD SECRET statement can add an Oracle Key Vault password to a TDE wallet.

Example 9-1 shows how to add an Oracle Key Vault password as a secret into an existing TDE wallet (for example, after migrating to Oracle Key Vault, the Oracle Key Vault password can be added to the old TDE wallet to set up an auto-open connection into Oracle Key Vault.

Example 9-1 Adding an Oracle Database Secret to a TDE wallet

```
ADMINISTER KEY MANAGEMENT
ADD SECRET 'external_key_manager_password' FOR CLIENT 'OKV_PASSWORD'
IDENTIFIED BY TDE_wallet_password WITH BACKUP;
```

Before migrating from a TDE wallet to Oracle Key Vault, you can upload the TDE wallet into the virtual wallet of that endpoint in Oracle Key Vault. After migrating, you now can delete the old TDE wallet, because its key is in Oracle Key Vault. In order to configure auto-open Oracle Key Vault without a wallet being already present, run the following statement:

```
ADMINISTER KEY MANAGEMENT ADD SECRET 'external_keystore_password'
FOR CLIENT 'OKV_PASSWORD' INTO [LOCAL] AUTO_LOGIN KEYSTORE 'WALLET_ROOT/tde';
```

Note that the setting TDE_CONFIGURATION='KEYSTORE_CONFIGUARTION=OKV' is for a password-protected connection into Oracle Key Vault. After the Oracle Key Vault password has been inserted into an existing or newly created wallet, change the TDE_CONFIGURATION setting to 'KEYSTORE_CONFIGURATION=OKV|FILE'.

9.1.12.4 Example: Changing an Oracle Key Vault Password Stored as a Secret in a TDE Wallet

The ADMINISTER KEY MANAGEMENT UPDATE SECRET statement can change an Oracle Key Vault password that is stored as a secret in a TDE wallet.

Example 9-2 shows how to change an Oracle Key Vault password that is stored as a secret in a TDE wallet.

Example 9-2 Changing an Oracle Key Vault Password Secret to a TDE Wallet

```
ADMINISTER KEY MANAGEMENT
UPDATE SECRET admin_password FOR CLIENT 'admin@myhost'
USING TAG 'new_host_credentials' FORCE KEYSTORE
IDENTIFIED BY TDE_wallet_password;
```

In this version, the password for the keystore is in an external store:

```
DMINISTER KEY MANAGEMENT
UPDATE SECRET admin_password FOR CLIENT 'admin@myhost'
USING TAG 'new_host_credentials' FORCE KEYSTORE
IDENTIFIED BY EXTERNAL STORE;
```

9.1.12.5 Example: Deleting an Oracle Key Vault Password Stored as a Secret in a TDE Wallet

The `ADMINISTER KEY MANAGEMENT DELETE SECRET` statement can delete Oracle Key Vault passwords that are stored as secrets in a TDE wallet.

Example 9-3 shows how to delete an Oracle Key Vault password that is stored as a secret in the TDE wallet.

Example 9-3 Deleting an Oracle Key Vault Password Secret in a TDE Wallet

```
ADMINISTER KEY MANAGEMENT
DELETE SECRET FOR CLIENT 'OKV_PASSWORD'
FORCE KEYSTORE
IDENTIFIED BY TDE_wallet_password WITH BACKUP;
```

In this version, the password for the TDE wallet is in an external store:

```
ADMINISTER KEY MANAGEMENT
DELETE SECRET FOR CLIENT 'OKV_PASSWORD'
FORCE KEYSTORE
IDENTIFIED BY EXTERNAL STORE WITH BACKUP;
```

9.1.12.6 Storing Oracle Database Secrets in an External Keystore in Isolated Mode

The `ADMINISTER KEY MANAGEMENT ADD SECRET|UPDATE SECRET|DELETE SECRET` statements can add secrets, update secrets, and delete secrets in a keystore.

1. Connect to the isolated mode PDB as a user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.
2. Add, update, or delete a database secret in an external keystore by using the following syntax:
   - To add a secret:

     ```
     ADMINISTER KEY MANAGEMENT
     ADD SECRET 'secret' FOR CLIENT 'client_identifier'
     [USING TAG 'tag']
     [TO [[LOCAL] AUTO_LOGIN] KEYSTORE keystore_location
     [FORCE KEYSTORE]
     IDENTIFIED BY "external_key_manager_password"]
     [WITH BACKUP [USING backup_id]];
     ```

   - To update a secret:

     ```
     ADMINISTER KEY MANAGEMENT
     UPDATE SECRET 'secret' FOR CLIENT 'client_identifier'
     [USING TAG 'tag']
      [TO [[LOCAL] AUTO_LOGIN] KEYSTORE keystore_location
     [FORCE KEYSTORE]
     IDENTIFIED BY "external_key_manager_password"
     [WITH BACKUP [USING backup_id]];
     ```

   - To delete a secret:

```
ADMINISTER KEY MANAGEMENT
DELETE SECRET FOR CLIENT 'client_identifier'
[FROM [[LOCAL] AUTO_LOGIN] KEYSTORE keystore_location
[FORCE KEYSTORE]
IDENTIFIED BY "eexternal_key_manager_password";
```

The specification is as follows:

- *secret* is the client secret key to be stored, updated, or deleted. To find information about existing secrets and their client identifiers, query the V$CLIENT_SECRETS dynamic view.
- *client_identifier* is an alphanumeric string used to identify the secret key.
- FORCE KEYSTORE temporarily opens the password-protected TDE wallet for this operation. You must open the TDE wallet for this operation.
- *external_key_manager_password* is for an external keystore manager, which can be Oracle Key Vault or OCI Vault - Key Management. Enclose this password in double quotation marks. For Oracle Key Vault, enter the password that was given during the Oracle Key Vault client installation. If at that time no password was given, then the password in the ADMINISTER KEY MANAGEMENT statement becomes NULL.

Related Topics

- Storing Oracle Database Secrets in Isolated Mode
  Secrets are data that support internal Oracle Database features that integrate external clients such as Oracle GoldenGate into the database.

9.1.12.7 Example: Adding an Oracle Database Secret to an External Keystore

The ADMINISTER KEY MANAGEMENT ADD SECRET statement can add an Oracle Database secret to an external keystore.

Example 9-4 shows how to add a password for a user to an external keystore.

Example 9-4 Adding an Oracle Database Secret to an External Keystore

```
ADMINISTER KEY MANAGEMENT ADD SECRET 'password'
FOR CLIENT 'admin@myhost' USING TAG 'myhost admin credentials'
IDENTIFIED BY "external_key_manager_password";
```

In this version, the keystore password is in an external store, so the EXTERNAL STORE setting is used for IDENTIFIED BY:

```
ADMINISTER KEY MANAGEMENT ADD SECRET 'password'
FOR CLIENT 'admin@myhost' USING TAG 'myhost admin credentials'
IDENTIFIED BY EXTERNAL STORE;
```

9.1.12.8 Example: Changing an Oracle Database Secret in an External Keystore

The ADMINISTER KEY MANAGEMENT MANAGEMENT UPDATE SECRET statement can change an Oracle Database secret in an external keystore.

Example 9-5 shows how to change a password that is stored as a secret in an external keystore.

Example 9-5 Changing an Oracle Database Secret in an External Keystore

```
ADMINISTER KEY MANAGEMENT MANAGEMENT UPDATE SECRET 'password2'
FOR CLIENT 'admin@myhost' USING TAG 'New host credentials'
IDENTIFIED BY "external_key_manager_password";
```

In this version, the password for the keystore is in an external store:

```
ADMINISTER KEY MANAGEMENT MANAGEMENT UPDATE SECRET 'password2'
FOR CLIENT 'admin@myhost' USING TAG 'New host credentials'
IDENTIFIED BY EXTERNAL STORE;
```

9.1.12.9 Example: Deleting an Oracle Database Secret in an External Keystore

The `ADMINISTER KEY MANAGEMENT DELETE SECRET FOR CLIENT` statement can delete an Oracle Database secret that is in an external keystore.

Example 9-6 shows how to delete an external key manager password that is stored as a secret in the external keystore.

Example 9-6 Deleting an Oracle Database Secret in an External Keystore

```
ADMINISTER KEY MANAGEMENT DELETE SECRET FOR CLIENT 'admin@myhost'
IDENTIFIED BY "external_key_manager_password";
```

In this version, the password for the keystore is in an external store:

```
ADMINISTER KEY MANAGEMENT DELETE SECRET FOR CLIENT 'admin@myhost'
IDENTIFIED BY EXTERNAL STORE;
```

9.1.13 Storing Oracle GoldenGate Secrets in a Keystore in Isolated Mode

You can store Oracle GoldenGate secrets in Transparent Data Encryption keystores.

9.1.13.1 About Storing Oracle GoldenGate Secrets in Keystores in Isolated Mode

You can use a keystore (TDE wallet or external keystore) to store secret keys for tools and external clients such as Oracle GoldenGate.

The secret key must be a string adhering to Oracle identifier rules. You can add, update, or delete a client secret in an existing keystore. This section describes how to capture Transparent Data Encryption encrypted data in the Oracle GoldenGate Extract (Extract) process using classic capture mode.

TDE support when Extract is in classic capture mode requires the exchange of the following keys:

- TDE support for Oracle GoldenGate in the classic capture mode of the Extract process requires that an Oracle database and the Extract process share the secret to encrypt sensitive information being exchanged. The shared secret is stored securely in the Oracle database and Oracle GoldenGate domains. The shared secret is stored in the TDE wallet or the external keystore as the database secret.
- The decryption key is a password known as the shared secret that is stored securely in the Oracle database and Oracle GoldenGate domains. Only a party that has possession of the shared secret can decrypt the table and redo log keys.

After you configure the shared secret, Oracle GoldenGate Extract uses the shared secret to decrypt the data. Oracle GoldenGate Extract does not handle the TDE master encryption key itself, nor is it aware of the keystore password. The TDE master encryption key and password remain within the Oracle database configuration.

Oracle GoldenGate Extract only writes the decrypted data to the Oracle GoldenGate trail file, which Oracle GoldenGate persists during transit. You can protect this file using your site's operating system standard security protocols, as well as the Oracle GoldenGate AES encryption options. Oracle GoldenGate does not write the encrypted data to a discard file

(specified with the `DISCARDFILE` parameter). The word `ENCRYPTED` will be written to any discard file that is in use.

Oracle GoldenGate does require that the keystore be open when processing encrypted data. There is no performance effect of Oracle GoldenGate feature on the TDE operations.

9.1.13.2 Oracle GoldenGate Extract Classic Capture Mode TDE Requirements

Ensure that you meet the requirements for Oracle GoldenGate Extract to support Transparent Data Encryption capture.

The requirements are as follows:

- To maintain high security standards, ensure that the Oracle GoldenGate Extract process runs as part of the Oracle user (the user that runs the Oracle database). That way, the keys are protected in memory by the same privileges as the Oracle user.
- Run the Oracle GoldenGate Extract process on the same computer as the Oracle database installation.

9.1.13.3 Configuring Keystore Support for Oracle GoldenGate

You can configure Transparent Data Encryption keystore support for Oracle GoldenGate by using a shared secret for the keystore.

9.1.13.3.1 Step 1: Decide on a Shared Secret for the Keystore

A shared secret for a keystore is a password.

- Decide on a shared secret that meets or exceeds Oracle Database password standards.

Do not share this password with any user other than trusted administrators who are responsible for configuring Transparent Data Encryption to work with Oracle GoldenGate Extract.

Related Topics

- *Oracle Database Security Guide*

9.1.13.3.2 Step 2: Configure Oracle Database for TDE Support for Oracle GoldenGate

The `DBMS_INTERNAL_CLKM` PL/SQL package enables you to configure TDE support for Oracle GoldenGate.

1. Connect to the united mode CDB root or isolated mode PDB as a user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.
2. Load the Oracle Database-supplied `DBMS_INTERNAL_CLKM` PL/SQL package.

   For example:

   ```
   @?/app/oracle/product/20.1/rdbms/admin/prvtclkm.plb
   ```

   The `prvtclkm.plb` file also enables Oracle GoldenGate to extract encrypted data from an Oracle database.
3. Grant the `EXECUTE` privilege on the `DBMS_INTERNAL_CLKM` PL/SQL package to the Oracle GoldenGate Extract database user.

   For example:

   ```
   GRANT EXECUTE ON DBMS_INTERNAL_CLKM TO psmith;
   ```

This procedure enables the Oracle database and Oracle GoldenGate Extract to exchange information.

9.1.13.3.3 Step 3: Store the TDE GoldenGate Shared Secret in the Keystore

The `ADMINISTER KEY MANAGEMENT` statement can store a TDE GoldenGate shared secret in a keystore.

Before you begin this procedure, ensure that you have configured the TDE software or external keystore.

1. Connect to the united mode CDB root or isolated mode PDB as a user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.
2. Set the Oracle GoldenGate-TDE key in the keystore by using the following syntax.

```
ADMINISTER KEY MANAGEMENT ADD|UPDATE|DELETE SECRET 'secret'
FOR CLIENT 'secret_identifier' [USING TAG 'tag']
IDENTIFIED BY keystore_password [WITH BACKUP [USING 'backup_identifier']];
```

   In this specification:

   - *secret* is the client secret key to be stored, updated, or deleted. Enclose this setting in single quotation marks (`' '`).
   - *secret_identifier* is an alphanumeric string used to identify the secret key. *secret_identifier* does not have a default value. Enclose this setting in single quotation marks (`' '`).
   - *tag* is an optional, user-defined description for the secret key to be stored. *tag* can be used with the `ADD` and `UPDATE` operations. Enclose this setting in single quotation marks (`' '`). This tag appears in the `SECRET_TAG` column of the `V$CLIENT_SECRETS` view.
   - *keystore_password* is the password for the keystore that is configured.
   - `WITH BACKUP` is required in case the keystore was not backed up before the `ADD`, `UPDATE` or `DELETE` operation. *backup_identifier* is an optional user-defined description for the backup. Enclose *backup_identifier* in single quotation marks (`' '`).

   The following example adds a secret key to the keystore and creates a backup in the same directory as the keystore:

```
ADMINISTER KEY MANAGEMENT ADD SECRET 'some_secret'
FOR CLIENT 'ORACLE_GG' USING TAG 'GoldenGate Secret'
IDENTIFIED BY password WITH BACKUP USING 'GG backup';
```

3. Verify the entry that you just created.

   For example:

```
SELECT CLIENT, SECRET_TAG FROM V$CLIENT_SECRETS WHERE CLIENT = 'ORACLEGG';

CLIENT   SECRET_TAG
-------- ----------------------------------------
ORACLEGG some_secret
```

4. Switch the log files.

```
CONNECT / AS SYSDBA

ALTER SYSTEM SWITCH LOGFILE;
```

9.1.13.3.4 Step 4: Set the TDE Oracle GoldenGate Shared Secret in the Extract Process

The GoldenGate Software Command Interface (GGSCI) utility sets the TDE Oracle GoldenGate shared secret in the extract process.

1. Start the GGSCI utility.

   ```
   ggsci
   ```

2. In the GGSCI utility, run the `ENCRYPT PASSWORD` command to encrypt the shared secret within the Oracle GoldenGate Extract parameter file.

   ```
   ENCRYPT PASSWORD shared_secret algorithm ENCRYPTKEY keyname
   ```

   In this specification:

   - *`shared_secret`* is the clear-text shared secret that you created when you decided on a shared secret for the keystore. This setting is case sensitive.
   - *`algorithm`* is `AES256`.
   - *`keyname`* is the logical name of the encryption key in the `ENCKEYS` lookup file. Oracle GoldenGate uses this name to look up the actual key in the `ENCKEYS` file.

   For example:

   ```
   ENCRYPT PASSWORD password AES256 ENCRYPTKEY mykey1
   ```

3. In the Oracle GoldenGate Extract parameter file, set the `DBOPTIONS` parameter with the `DECRYPTPASSWORD` option.

   As input, supply the encrypted shared secret and the Oracle GoldenGate-generated or user-defined decryption key.

   ```
   DBOPTIONS DECRYPTPASSWORD shared_secret algorithm ENCRYPTKEY keyname
   ```

   In this specification:

   - *`shared_secret`* is the clear-text shared secret that you created when you decided on a shared secret for the keystore. This setting is case sensitive.
   - *`algorithm`* is `AES256`.
   - *`keyname`* is the logical name of the encryption key in the `ENCKEYS` lookup file.

     For example:

     ```
     DBOPTIONS DECRYPTPASSWORD AACAAAAAAAAAAAIALCKDZIRHOJBHOJUH AES256 ENCRYPTKEY
     mykey1
     ```

9.1.14 Migrating Keystores in Isolated Mode

You can perform migration and reverse migration operations between TDE wallets and external keystores in isolated mode.

9.1.14.1 Reverse Migrating an Isolated PDB from Oracle Key Vault to a TDE Wallet

Isolated PDBs have individual keystores (TDE wallets or Oracle Key Vault external keystores), and can individually be migrated from the TDE wallet to Oracle Key Vault, and individually reverse migrated from Oracle Key Vault back to the TDE wallet.

For both the TDE wallet and the external keystore to open at the same time, either the TDE wallet must have the same password as the external keystore, or alternatively, after the migration has completed you can create an auto-login keystore for the TDE wallet.

1. Before migrating any database from wallet to Oracle Key Vault, upload the contents of the wallet (the current and retired TDE master encryption keys) into the virtual wallet that you created in Oracle Key Vault for that database.

   In the following example, the static initialization parameter `WALLET_ROOT` is set to `/etc/ORACLE/KEYSTORES/${ORACLE_SID}`:

   ```
   /etc/ORACLE/KEYSTORES/${ORACLE_SID}/pdb-guid/okv/bin/okvutil upload -l
   /etc/ORACLE/KEYSTORES/${ORACLE_SID}/pdb-guid/tde/ -t wallet -g
   virtual_wallet_name_in_Oracle_Key_Vault -v 2
   ```

2. Connect to the isolated mode PDB as a user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.

3. Set the password of the external keystore so that it matches that of the TDE wallet.

   Some Oracle tools, such as Oracle Data Pump and Oracle Recovery Manager, require access to the old TDE wallet to decrypt data that was exported or backed up using the TDE master encryption key from the old TDE wallet.

   - To set the TDE wallet password so that it is the same as that of the external keystore, use the following syntax:

     ```
     ADMINISTER KEY MANAGEMENT ALTER KEYSTORE PASSWORD
     [FORCE KEYSTORE]
     IDENTIFIED BY TDE_wallet_password
     SET "external_key_manager_password"
     WITH BACKUP [USING 'backup_identifier'];
     ```

     In this specification:

     - *TDE_wallet_password* is the password that was assigned to this wallet when it was created.
     - *external_key_manager_password* is for an external keystore manager, which can be Oracle Key Vault or OCI Vault - Key Management. Enclose this password in double quotation marks. For Oracle Key Vault, enter the password that was given during the Oracle Key Vault client installation. If at that time no password was given, then the password in the `ADMINISTER KEY MANAGEMENT` statement becomes `NULL`.

   - Alternatively, to create an auto-login keystore for a TDE wallet, use the following syntax:

     ```
     ADMINISTER KEY MANAGEMENT CREATE [LOCAL] AUTO_LOGIN KEYSTORE
     FROM KEYSTORE 'keystore_location'
     IDENTIFIED BY TDE_wallet_password;
     ```

4. Provision Oracle Key Vault for the isolated mode PDB by following the instructions in *Oracle Key Vault Administrator's GuideOracle Key Vault Administrator's Guide* to install the Oracle Key Vault software onto the endpoint.

5. Set the configuration of the keystore so that the external keystore becomes the new primary, and the password-protected TDE wallet becomes the secondary, as follows:

```
ALTER SYSTEM SET TDE_CONFIGURATION="KEYSTORE_CONFIGURATION=OKV|FILE";
```

6. Migrate the external keystore by using the following syntax:

```
ADMINISTER KEY MANAGEMENT SET ENCRYPTION KEY
IDENTIFIED BY "external_key_manager_password"
[FORCE KEYSTORE]
MIGRATE USING TDE_wallet_password;
```

After you complete the migration, you do not need to restart the database, nor do you need to manually re-open the external keystore.

Related Topics

- Keystore Order After a Migration
  After you perform a migration, keystores can be either primary or secondary in their order.
- Migration of Keystores to and from Oracle Key Vault
  You can use Oracle Key Vault to migrate both TDE wallets and external keystores to and from Oracle Key Vault.

9.1.14.2 Migrating from an External Keystore to a Password-Protected TDE Wallet in Isolated Mode

In isolated mode, you can migrate from an external keystore to a password-protected TDE wallet.

1. Connect to the isolated mode PDB as a user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.
2. Ensure that you have enrolled the PDB endpoint and that the Oracle Key Vault configuration is present at the *$WALLET_ROOT/pdb_guid*`/okv` location.

   For both the TDE wallet and the external keystore to open at the same time, either the TDE wallet must have the same password as the external keystore, or alternatively, after the reverse migration has completed, you can create an auto-login TDE wallet.
3. Set the `TDE_CONFIGURATION` parameter as follows, so that `FILE` becomes the new primary keystore, and `OKV` becomes the secondary keystore.

```
ALTER SYSTEM SET TDE_CONFIGURATION="KEYSTORE_CONFIGURATION=FILE|OKV";
```

4. Now that the keystore configuration has been completed, issue the following statement to reverse migrate from the external keystore to the password-protected TDE wallet:

```
ADMINISTER KEY MANAGEMENT SET ENCRYPTION KEY
IDENTIFIED BY TDE_wallet_password
REVERSE MIGRATE USING "external_key_manager_password"
[WITH BACKUP [USING 'backup_identifier']];
```

5. Optionally, change the password of the newly migrated TDE wallet.

   For example:

```
ADMINISTER KEY MANAGEMENT ALTER KEYSTORE PASSWORD
IDENTIFIED BY
old_password SET new_password
WITH BACKUP USING 'pwd_change';
```

After you complete these steps, the migration process automatically reloads the TDE wallet keys in memory. You do not need to restart the database, nor do you need to manually re-open

the TDE wallet. The external keystore may still be required after reverse migration because the old keys are likely to have been used for encrypted backups or by tools such as Oracle Data Pump and Oracle Recovery Manager. You should create an auto-login TDE wallet and put the `HSM_PASSWORD` client secret into it. For example:

```
ADMINISTER KEY MANAGEMENT ADD SECRET 'external_key_manager_password'
FOR CLIENT 'EXTERNAL_PASSWORD'
TO LOCAL AUTO_LOGIN KEYSTORE TDE_wallet_location
WITH BACKUP;
```

Related Topics

- About Migrating Back from an External Keystore
  To switch from using an external keystore solution to a TDE wallet, you can use reverse migration of the TDE wallet.
- Keystore Order After a Migration
  After you perform a migration, keystores can be either primary or secondary in their order.
- Migration of Keystores to and from Oracle Key Vault
  You can use Oracle Key Vault to migrate both TDE wallets and external keystores to and from Oracle Key Vault.

9.1.15 Uniting a Pluggable Database Keystore

Uniting a PDB keystore moves the TDE master encryption keys from the PDB keystore into the keystore of the CDB root. This enables the administrator of the keystore of the CDB root to manage the keys.

The client secrets are not moved. Instead, they are left behind in the keystore that the PDB used while it was configured in isolated mode. Oracle recommends that you delete client secrets from that keystore before you unite the PDB keystore. Similarly, when a PDB becomes isolated, no client secret contained in the keystore of the CDB root is moved.

1. Connect to the isolated mode PDB as a user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.
2. Unite the PDB keystore, which moves the TDE master encryption keys from the PDB keystore into the keystore of the CDB root, by using the following syntax:

   ```
   ADMINISTER KEY MANAGEMENT UNITE KEYSTORE
   IDENTIFIED BY isolated_keystore_password
   WITH ROOT KEYSTORE [FORCE KEYSTORE]
   IDENTIFIED BY
   [EXTERNAL STORE | keystore_password_of_cdb_root]
   [WITH BACKUP [USING backup_id]];
   ```

   In this specification:

   - `FORCE KEYSTORE` temporarily opens the password-protected TDE wallet for this operation if an auto-login TDE wallet is open (and in use) or if the TDE wallet is closed.
   - *`united_keystore_password`*: Knowledge of this password does not enable the user who performs the `UNITE KEYSTORE` operation privileges to perform `ADMINISTER KEY MANAGEMENT UNITE KEYSTORE` operations on the PDB.

   When the keystore of a PDB is united with a keystore in the CDB root, all of the previously active (historical) TDE master encryption keys that were associated with the PDB are moved to the keystore of the CDB root. (If an `ORA-46694: The keys are already in the root keystore` error appears, see below.)
3. Confirm that the isolated mode PDB is now a united mode PDB.

```
SELECT KEYSTORE_MODE FROM V$ENCRYPTION_WALLET;
```

The output should be `UNITED`.

The keystore no longer exists but its master encryption key is now in the keystore in the CDB root. If you later decide that you want the united mode PDB to be an isolated mode PDB again, then you can use the `ADMINISTER KEY MANAGEMENT ISOLATE KEYSTORE` statement.

ORA-46694 error: If a wallet is created in the in a PDB context, then it changes the keystore type of the PDB to isolated, and unless a key is set for the PDB, the wallet status will be `OPEN_NO_MASTER_KEY`. Uniting this type of PDB using the `ADMINISTER KEY MANAGEMENT UNITE KEYSTORE` statement will result in an `ORA-46694: The keys are already in the root keystore` error. To change the `KEYSTORE_MODE` to `UNITED` for this PDB, you must change the TDE configuration for the PDB. In the PDB, run the following statement:

```
ALTER SYSTEM RESET TDE_CONFIGURATION;
```

Related Topics

- Isolating a Pluggable Database Keystore
  Isolating a PDB keystore moves the master encryption key from the CDB root keystore into an isolated mode keystore in the a PDB.

9.1.16 Creating a Keystore When the PDB Is Closed

When you create a keystore in a PDB that is closed, the new keystore is empty and the PDB is converted to isolated mode.

9.1.16.1 About Creating a Keystore When the PDB Is Closed

Creating a keystore in a PDB that is closed could inadvertently cause problems in rekey operations, but the keystore creation can be reverted.

In previous releases, if you tried to create a keystore in a closed PDB, you were prevented and an `ORA-65040: operation not allowed from within a pluggable database` error would appear. Starting in Oracle Database release 18c, for convenience, when the keystore of the PDB is closed and if you run the `ADMINISTER KEY MANAGEMENT CREATE KEYSTORE` statement in the PDB, Oracle Database allows the operation.

If the closed PDB has not been configured to use encryption (that is, it has never had an `ADMINISTER KEY MANAGEMENT SET KEY` statement performed in it), after you run `ADMINISTER KEY MANAGEMENT CREATE KEYSTORE`, resulting in an empty keystore and the configuration of the PDB being changed to isolated mode, then you can create a TDE master encryption key in this empty keystore.

If, however, the PDB was already configured to use encryption, then the PDB may be configured in united mode (and thus have its TDE master encryption key being managed in the keystore of the CDB root).

Mistakenly running an `ADMINISTER KEY MANAGEMENT CREATE KEYSTORE` statement on such a closed PDB will create an additional keystore (which will be empty), and will then configure the PDB to be in isolated mode. This effectively misconfigures the PDB, because the PDB is now in isolated mode (whereas it should be in united mode), yet its TDE master encryption key is still in the keystore of the CDB root. This misconfiguration can cause problems later on, if you try to rekey the TDE master encryption key by using the `ADMINISTER KEY MANAGEMENT SET ENCRYPTION KEY` statement . An `ORA-28362: master key not found` error will appear, because when encryption has already been enabled and a key has been set, Oracle Database treats

the `ADMINISTER KEY MANAGEMENT SET ENCRYPTION KEY` statement as a rekey operation. In order to perform a rekey operation, Oracle Database must locate the currently active TDE master encryption key of the PDB. But in this misconfigured PDB, Oracle Database cannot locate the TDE master encryption key, because the PDB is now in isolated mode and the necessary key is in the keystore of the CDB root. Hence, the PDB is no longer configured to search in the keystore of the PDB, and the rekey operation fails.

To remedy the misconfiguration of the PDB, you must reconfigure the PDB to united mode and you must remove the empty keystore. (Always make a backup before removing any keystore.) When the PDB is configured back to united mode, then the currently active TDE master encryption key is once again available for rekey and other TDE master encryption key operations.

If later on you want to configure the PDB to be in isolated mode, then you can open the PDB and run the `ADMINISTER KEY MANAGEMENT ISOLATE KEYSTORE` statement, which isolates the PDB and moves its TDE master encryption key and previously-active (historical) keys from the keystore of the CDB root to a newly-created keystore for the isolated PDB.

Related Topics

- Reverting a Keystore Creation Operation When a PDB Is Closed
  If you have inadvertently created a keystore in a PDB (and thereby caused it to become configured in isolated mode), then you should reverse the keystore creation operation.

9.1.16.2 Reverting a Keystore Creation Operation When a PDB Is Closed

If you have inadvertently created a keystore in a PDB (and thereby caused it to become configured in isolated mode), then you should reverse the keystore creation operation.

Use this procedure if you created a keystore in a closed PDB that already had encryption enabled (that is, it already had a TDE master encryption key).

1. Connect to the isolated mode PDB as a user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.
2. Confirm the mode of the PDB by querying the `KEYSTORE_MODE` column of the `V$ENCRYPTION_WALLET` dynamic view.
3. If the `V$ENCRYPTION_WALLET` output is `ISOLATED`, then run the `ALTER SYSTEM` statement to reconfigure the PDB to united mode.
   - When pfile is in use, clear the `TDE_CONFIGURATION` parameter by using the following statement:

     ```
     ALTER SYSTEM RESET TDE_CONFIGURATION SCOPE=MEMORY;
     ```

     In an Oracle Real Application Clusters environment, include the `SID` parameter:

     ```
     ALTER SYSTEM RESET TDE_CONFIGURATION SCOPE=MEMORY SID='*';
     ```

   - When spfile is in use, clear the `TDE_CONFIGURATION` parameter by using this statement:

     ```
     ALTER SYSTEM RESET TDE_CONFIGURATION SCOPE=BOTH;
     ```

     In an Oracle Real Application Clusters environment, include the `SID` parameter:

     ```
     ALTER SYSTEM RESET TDE_CONFIGURATION SCOPE=BOTH SID='*';
     ```

4. In the *`WALLET_ROOT`*`/`*`pdb_guid`*`/tde` directory, find and back up the `ewallet.p12` keystore file that was mistakenly created.
5. Delete the mistakenly-created empty keystore file.

At this stage, the PDB will be in united mode and the correct keystore and TDE master encryption key will be available for any future rekey operations.

9.2 Administering Transparent Data Encryption in Isolated Mode

You can perform a number of general administrative tasks with Transparent Data Encryption in isolated mode.

9.2.1 Cloning or Relocating Encrypted PDBs in Isolated Mode

You can clone or relocate encrypted PDBs within the same container database, or across container databases.

If you are trying to move a PDB in which the `SYSTEM`, `SYSAUX`, `UNDO`, or `TEMP` tablespace is encrypted, and using the manual export or import of keys, then you must first import the keys for the PDB in the target database's `CDB$ROOT` before you create the PDB. Import of the keys are again required inside the PDB to associate the keys to the PDB.

- Clone or relocate the PDB using the following syntax:

```
CREATE|RELOCATE PLUGGABLE DATABASE database_name KEYSTORE
IDENTIFIED BY EXTERNAL STORE|target_keystore_password [NO REKEY];
```

Related Topics

- *Oracle Multitenant Administrator's Guide*

9.2.2 Unplugging and Plugging a PDB with Encrypted Data in a CDB in Isolated Mode

In isolated mode, for a PDB that has encrypted data, you can plug it into a CDB. Conversely, you can unplug this PDB from the CDB.

9.2.2.1 Unplugging a PDB That Has Encrypted Data in Isolated Mode

You can unplug a PDB (that has encrypted data) from one CDB and then optionally plug it into another CDB.

Unlike united mode, you do not need to specify the `ENCRYPT` clause in the `ALTER PLUGGABLE DATABASE` statement. The database that is unplugged contains data files and other associated files. Because each PDB can have its own unique keystore, you do not need to export the TDE master encryption key of the PDB that you want to unplug. You can check if a PDB has already been unplugged by querying the `STATUS` column of the `DBA_PDBS` data dictionary view.

- Unplug the isolated mode PDB as you normally unplug PDBs.

  For example:

```
ALTER PLUGGABLE DATABASE pdb1
  UNPLUG INTO '/oracle/data/pdb1.xml';
```

Related Topics

- *Oracle Multitenant Administrator's Guide*
- *Oracle Database SQL Language Reference*

9.2.2.2 Plugging a PDB That Has Encrypted Data into a CDB in Isolated Mode

After you plug a PDB that has encrypted data into a CDB, you can set the encryption key in the PDB.

Unlike united mode, you do not need to specify the `DECRYPT` clause in the `CREATE PLUGGABLE DATABASE` statement. When you plug an unplugged PDB into another CDB, the key version is set to `0` because this operation invalidates the history of the previous keys. You can check the key version by querying the `KEY_VERSION` column of the `V$ENCRYPTED_TABLESPACES` dynamic view. Similarly, if a control file is lost and recreated, then the previous history of the keys is reset to `0`. You can check if a PDB has already been plugged in by querying the `STATUS` column of the `DBA_PDBS` data dictionary view.

1. Create the PDB by plugging the unplugged PDB into the CDB.
   - For example, if you had exported the PDB data into a metadata XML file:

     ```
     CREATE PLUGGABLE DATABASE CDB1_PDB2
     USING '/tmp/cdb1_pdb2.xml'
     NOCOPY KEYSTORE
     IDENTIFIED BY password;
     ```

   - If you had exported the PDB into an archive file:

     ```
     CREATE PLUGGABLE DATABASE CDB1_PDB2
     USING '/tmp/cdb1_pdb2.pdb';
     ```

   During the open operation of the PDB after the plug operation, Oracle Database determines if the PDB has encrypted data. If so, it opens the PDB in the `RESTRICTED` mode.

   You can find if the source database has encrypted data or a TDE master encryption key set in the keystore by querying the `V$ENCRYPTION_KEYS` dynamic view.
2. Open the PDB.

   For example:

   ```
   ALTER PLUGGABLE DATABASE CDB1_PDB2 OPEN;
   ```

3. Open the keystore in the CDB root.

   For example:

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

   Optionally, open the keystore in the PDB.
4. In the PDB, open the keystore and set the TDE master encryption key for the PDB.

   For example:

   ```
   ADMINISTER KEY MANAGEMENT SET KEY
   IDENTIFIED BY keystore_password
   WITH BACKUP USING 'emp_key_backup';
   ```

9.2.2.3 Unplugging a PDB That Has Master Encryption Keys Stored in an External Keystore in Isolated Mode

You can unplug a PDB from one CDB that has been configured with an external keystore and then plug it into another CDB also configured with an external keystore.

1. Unplug the PDB.

You can check if a PDB has already been unplugged by querying the `STATUS` column of the `DBA_PDBS` data dictionary view.

2. Move the master encryption keys of the unplugged PDB from the hardware that was used at the source CDB to the hardware that is in use at the destination CDB.

   Refer to the documentation for the external keystore for information about moving master keys between external keystores.

Related Topics

- *Oracle Multitenant Administrator's Guide*

9.2.2.4 Plugging a PDB That Has Master Keys Stored in an External Keystore in Isolated Mode

The `ADMINISTER KEY MANAGEMENT` statement can import an external keystore master encryption key to a PDB that has been moved to another CDB.

1. Plug the unplugged isolated mode PDB into the destination CDB that has been configured with the external keystore.

   You can check if a PDB has already been plugged in by querying the `STATUS` column of the `DBA_PDBS` data dictionary view.

   After the plug-in operation, the PDB that has been plugged in will be in restricted mode.

2. Ensure that the master keys from the external keystore that has been configured with the source CDB are available in the external keystore of the destination CDB.
3. Connect to the plugged PDB as a user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.
4. Open the keystore of the plugged PDB.

   For example, for a PDB called `PDB1`:

   ```
   ALTER SESSION SET CONTAINER = PDB1;
   ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN
   IDENTIFIED BY "external_key_manager_password";
   ```

5. Import the external keystore master encryption key into the PDB.

   ```
   ADMINISTER KEY MANAGEMENT IMPORT ENCRYPTION KEYS
   WITH SECRET "external_keystore" FROM 'external_keystore'
   IDENTIFIED BY "external_key_manager_password";
   ```

6. Close and re-open the PDB.

   ```
   ALTER PLUGGABLE DATABASE PDB1 CLOSE;
   ALTER PLUGGABLE DATABASE PDB1 OPEN;
   ```

Related Topics

- *Oracle Multitenant Administrator's Guide*

9.2.3 Cloning a PDB with Encrypted Data in a CDB in Isolated Mode

The `CREATE PLUGGABLE DATABASE` statement with the `KEYSTORE IDENTIFIED BY` clause can clone a PDB that has encrypted data.

1. Connect to the isolated mode PDB as a user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.

2. Ensure that the TDE wallet of the PDB that you plan to clone is open.

   You can query the `STATUS` column of the `V$ENCRYPTION_WALLET` view to find if the TDE wallet is open.

   For example:

   ```
   ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN FORCE KEYSTORE IDENTIFIED BY
   TDE_wallet_password;
   ```

3. Clone the PDB.

   For example:

   ```
   CREATE PLUGGABLE DATABASE cdb1_pdb3 FROM cdb1_pdb1
   FILE_NAME_CONVERT=('cdb1_pdb1', 'pdb3/cdb1_pdb3') KEYSTORE
   IDENTIFIED BY TDE_wallet_password;
   ```

   Replace *TDE_wallet_password* with the password of the TDE wallet of the CDB where the `cdb1_pdb3` clone is created.

   After you create the cloned PDB, encrypted data is still accessible by the clone using the master encryption key of the original PDB. After a PDB is cloned, there may be user data in the encrypted tablespaces. This encrypted data is still accessible because the master encryption key of the source PDB is copied over to the destination PDB. Because the clone is a copy of the source PDB but will eventually follow its own course and have its own data and security policies, you should rekey the master encryption key of the cloned PDB.

4. Rekey the master encryption key of the cloned PDB.

   For example:

   ```
   ADMINISTER KEY MANAGEMENT SET KEY
   IDENTIFIED BY TDE_wallet_password
   WITH BACKUP USING 'emp_key_backup';
   ```

   Before you rekey the master encryption key of the cloned PDB, the clone can still use master encryption keys that belong to the original PDB. However, these master encryption keys do not appear in the cloned PDB `V$` dynamic views. Rekeying the master encryption key ensures that the cloned PDB uses its own unique keys, which will be viewable in the `V$` views.

Related Topics

- About Managing Cloned PDBs That Have Encrypted Data in United Mode
  When you clone a PDB, you must make the master encryption key of the source PDB available to cloned PDB.

9.2.4 Remotely Cloning an Encrypted PDB in Isolated Mode

The `CREATE PLUGGABLE DATABASE` statement with the `KEYSTORE IDENTIFIED BY` clause can remotely clone a PDB that has encrypted data.

1. Connect to the isolated mode PDB as a user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.
2. Query the `STATUS` column of the `V$ENCRYPTION_WALLET` view to ensure that the TDE wallet of the PDB that you plan to clone is open.
3. Create a database link for the PDB that you want to clone remotely.

Use the `CREATE DATABASE LINK` SQL statement to create the database link. You must create the database link by following the database link prerequisites that are required for cloning a remote PDB.

4. Use the `CREATE PLUGGABLE DATABASE` statement with the `KEYSTORE IDENTIFIED BY` clause to perform the clone of the PDB.

   For example:

   ```
   CREATE PLUGGABLE DATABASE target_pdb
   FROM source_pdb@clone_link KEYSTORE
   IDENTIFIED BY TDE_wallet_password;
   ```

   The *TDE_wallet_password* is the password of a new TDE wallet that will be created in *WALLET_ROOT*/*target_pdb_guid*/`tde` for the incoming PDB.

   After you create the cloned PDB, encrypted data is still accessible because the master encryption key of the source PDB is copied over to the destination PDB. Because it will eventually follow its own course and have its own data and security policies, you should rekey the master encryption key of the cloned PDB.

5. Rekey the master encryption key of the remotely cloned PDB.

   Run the following SQL statement to create an administer key management command with a `TAG`:

   ```
   SELECT ' ADMINISTER KEY MANAGEMENT SET KEY
   USING TAG '''||SYS_CONTEXT('USERENV', 'CON_NAME')||' '||TO_CHAR (SYSDATE,
   'YYYY-MM-DD HH24:MI:SS')||'''
   IDENTIFIED BY KEYSTORE_PASSWORD WITH BACKUP;'
   AS "SET KEY COMMAND" FROM DUAL;


   ADMINISTER KEY MANAGEMENT SET KEY USING TAG 'PDB_NAME DATE TIME'
   IDENTIFIED BY KEYSTORE_PASSWORD WITH BACKUP;
   ```

   Before you rekey the master encryption key of the cloned PDB, the clone can still use master encryption keys that belong to the original PDB. However, these master encryption keys do not appear in the cloned PDB `V$` dynamic views. Rekeying the master encryption key ensures that the cloned PDB uses its own unique keys, which will be viewable in the `V$` views.

Related Topics

- About Managing Cloned PDBs That Have Encrypted Data in United Mode
  When you clone a PDB, you must make the master encryption key of the source PDB available to cloned PDB.
- Opening the TDE Wallet in a United Mode PDB
  To open a TDE wallet in united mode, you must use the `ADMINISTER KEY MANAGEMENT` statement with the `SET KEYSTORE OPEN` clause.
- *Oracle Multitenant Administrator's Guide*
- *Oracle Multitenant Administrator's Guide*
- *Oracle Database SQL Language Reference*

9.2.5 Relocating an Encrypted PDB in Isolated Mode

The `CREATE PLUGGABLE DATABASE` statement with the `KEYSTORE IDENTIFIED BY` clause can relocate across CDBs a cloned PDB that has encrypted data.

1. Connect to the isolated mode PDB as a user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.
2. Query the `STATUS` column of the `V$ENCRYPTION_WALLET` view to ensure that the TDE wallet of the PDB that you want to relocate is open.
3. Create a database link for the PDB that you want to relocate.

   Use the `CREATE DATABASE LINK` SQL statement to create the database link. You must create the database link by following the database link prerequisites that are required for relocating a remote PDB or a non-CDB.
4. Use the `CREATE PLUGGABLE DATABASE` statement with the `KEYSTORE IDENTIFIED BY` clause to relocate the PDB.

   For example:

   ```
   CREATE PLUGGABLE DATABASE target_pdb_name
   FROM src_pdb_name@dblink RELOCATE
   KEYSTORE IDENTIFIED BY TDE_wallet_password;
   ```

   Replace *`TDE_wallet_password`* with the password of the TDE wallet of the CDB where the `cdb1_pdb3` clone is created.

   After you create the cloned PDB, encrypted data is still accessible by the clone using the master encryption key of the original PDB. After a PDB is cloned, there may be user data in the encrypted tablespaces. This encrypted data is still accessible because the master encryption key of the source PDB is copied over to the destination PDB. Because the clone is a copy of the source PDB but will eventually follow its own course and have its own data and security policies, you should rekey the master encrytion key of the cloned PDB.
5. Rekey the master encryption key of the remotely cloned PDB.

   For example:

   ```
   ADMINISTER KEY MANAGEMENT SET KEY
   FORCE KEYSTORE
   IDENTIFIED BY TDE_wallet_password
   WITH BACKUP USING 'emp_key_backup';
   ```

   In this example, `FORCE KEYSTORE` is included because the TDE wallet must be open during the rekey operation.

   Before you rekey the master encryption key of the cloned PDB, the clone can still use master encryption keys that belong to the original PDB. However, these master encryption keys do not appear in the cloned PDB `V$` dynamic views. Rekeying the master encryption key ensures that the cloned PDB uses its own unique keys, which will be viewable in the `V$` views.

Related Topics

- About Managing Cloned PDBs That Have Encrypted Data in United Mode
  When you clone a PDB, you must make the master encryption key of the source PDB available to cloned PDB.

- Opening the TDE Wallet in a United Mode PDB
  To open a TDE wallet in united mode, you must use the `ADMINISTER KEY MANAGEMENT` statement with the `SET KEYSTORE OPEN` clause.
- *Oracle Multitenant Administrator's Guide*
- *Oracle Multitenant Administrator's Guide*
- *Oracle Database SQL Language Reference*

9.2.6 How Keystore Open and Close Operations Work in Isolated Mode

You should be aware of how keystore (TDE wallets or external keystores) open and close operations work in isolated mode.

For each PDB in isolated mode, you must explicitly open the password-protected TDE wallet or external keystore in the PDB to enable the Transparent Data Encryption operations to proceed. (Auto-login and local auto-login TDE wallets open automatically.) Closing a keystore on a PDB blocks all of the Transparent Data Encryption operations on that PDB.

The open and close keystore operations in a PDB depend on the open and close status of the keystore in the PDB.

Note the following:

- You can create a separate keystore password for each PDB in the multitenant environment.
- Before you can manually open a password-protected software or an external keystore in an individual PDB, you must open the keystore in the CDB root.
- If an auto-login TDE wallet is in use, or if the TDE wallet is closed, then include the `FORCE KEYSTORE` clause in the `ADMINISTER KEY MANAGEMENT` statement when you open the TDE wallet.
- If the keystore is a password-protected TDE wallet that uses an external store for passwords, then replace the password in the `IDENTIFIED BY` clause with `EXTERNAL STORE`.
- Before you can set a TDE master encryption key in an individual PDB, you must set the key in the CDB root. Oracle highly recommends that you include the `USING TAG` clause when you set keys in PDBs. For example:

  ```
  SELECT ' ADMINISTER KEY MANAGEMENT SET KEY
  USING TAG '''||SYS_CONTEXT('USERENV', 'CON_NAME')||' '||TO_CHAR (SYSDATE,
  'YYYY-MM-DD HH24:MI:SS')||'''
  FORCE KEYSTORE IDENTIFIED BY EXTERNAL STORE
  WITH BACKUP CONTAINER = CURRENT;' AS "SET KEY COMMAND" FROM DUAL;
  ```

  Including the `USING TAG` clause enables you to quickly and easily identify the keys that belong to a certain PDB, and when they were created.
- Auto-login and local auto-login TDE wallets open automatically. You do not need to manually open these from the root first, or from the PDB.
- If there is any PDB configured in isolated mode that has its keystore open, then an attempt to close the keystore in the CDB root would fail with an `ORA-46692 cannot close wallet` error. Use the `FORCE CLOSE` clause in the `ADMINISTER KEY MANAGEMENT` statement to override this behavior.
- If you perform an `ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN` statement in the CDB root and set the `CONTAINER` clause to `ALL`, then the keystore will only be opened in each

open PDB that is configured in united mode. Keystores for any PDBs that are configured in isolated mode are not opened.

9.2.7 Exporting and Importing Master Encryption Keys for a PDB in Isolated Mode

In isolated mode, the `EXPORT` and `IMPORT` clauses of `ADMINISTER KEY MANAGEMENT EXPORT` can export or import master encryption keys for a PDB.

9.2.7.1 About Exporting and Importing Master Encryption Keys for a PDB in Isolated Mode

In isolated mode, you can export and import master encryption keys from the CDB root.

You can export and import all of the master encryption keys that belong to the PDB by exporting and importing the master encryption keys from within a PDB. Export and import operations of master encryption keys in a PDB supports the PDB unplug and plug operations. During a PDB unplug and plug operations, all the master encryption keys that belong to a PDB, as well as the metadata, are involved. Therefore, the `WITH IDENTIFIER` clause of the `ADMINISTER KEY MANAGEMENT EXPORT` statement is not allowed when you export keys from within a PDB. The `WITH IDENTIFIER` clause is only permitted in the CDB root.

You should include the `FORCE KEYSTORE` clause if the CDB root has an auto-login keystore or if the keystore is closed. If the keystore has been configured to use an external store for the password, then use the `IDENTIFIED BY EXTERNAL STORE` clause. For example, to perform an export operation for this scenario:

```
ADMINISTER KEY MANAGEMENT EXPORT KEYS
WITH SECRET "my_secret"
TO '/etc/TDE/export.exp'
FORCE KEYSTORE
IDENTIFIED BY EXTERNAL STORE;
```

This `ADMINISTER KEY MANAGEMENT EXPORT` operation exports not only the keys but creates metadata that is necessary for PDB environments (as well as for cloning operations).

Inside a PDB, the export operation of master encryption keys exports the keys that were created or activated by a PDB with the same GUID as the PDB where the keys are being exported. Essentially, all of the keys that belong to a PDB where the export is being performed will be exported.

The importing of master encryption keys from an export file within a PDB takes place only if the master encryption key was exported from another PDB with the same GUID. To support the plug-in of a non-CDB as PDB into a CDB, you must have already exported the TDE master encryption keys from the non-CDB and imported them into the PDB without the `WITH IDENTIFIER` clause. Because the PDB-specific details, such as the PDB name and database ID, can change from one CDB to the next, the PDB-specific information is modified during the import to reflect the updated PDB information.

> **Note:**
>
> Within a PDB, you can only export the keys of a PDB as a whole. The ability to export them selectively based on a query or an identifier is restricted to the root.

9.2.7.2 Exporting or Importing a Master Encryption Key for a PDB in Isolated Mode

In isolated mode, the `ADMINISTER KEY MANAGEMENT` statement can export or import a master encryption key for a PDB.

1. Connect to the isolated mode PDB as a user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege.
2. Perform the export or import operation.

   For example:

   ```
   ADMINISTER KEY MANAGEMENT EXPORT ENCRYPTION KEYS
   WITH SECRET "hr_secret" TO '/tmp/export.p12'
   FORCE KEYSTORE
   IDENTIFIED BY password;
   ```

   Ensure that you include the `FORCE KEYSTORE` clause because the keystore must be open for this operation.

Related Topics

- Example: Exporting a Master Encryption Key from a PDB in Isolated Mode
  The `ADMINISTER KEY MANAGEMENT EXPORT ENCRYPTION KEYS` statement can export master encryption keys for a PDB.

9.2.7.3 Example: Exporting a Master Encryption Key from a PDB in Isolated Mode

The `ADMINISTER KEY MANAGEMENT EXPORT ENCRYPTION KEYS` statement can export master encryption keys for a PDB.

Example 9-7 shows how to export a master encryption key from the PDB `hrpdb`. In this example, the `FORCE KEYSTORE` clause is included in case the auto-login keystore is in use, or if the keystore is closed.

Example 9-7 Exporting a Master Encryption Key from a PDB

```
ADMINISTER KEY MANAGEMENT EXPORT ENCRYPTION KEYS
WITH SECRET "my_secret" TO '/tmp/export.p12'
FORCE KEYSTORE
IDENTIFIED BY password;
```

9.2.7.4 Example: Importing a Master Encryption Key into a PDB in Isolated Mode

The `ADMINISTER KEY MANAGEMENT IMPORT ENCRYPTION KEYS` statement can import a master encryption key into a PDB.

Example 9-8 shows how to import a master encryption key into the PDB `hrpdb`.

Example 9-8 Importing a Master Encryption Key into a PDB

```
ADMINISTER KEY MANAGEMENT IMPORT ENCRYPTION KEYS
WITH SECRET "my_secret"
FROM '/tmp/export.p12'
FORCE KEYSTORE
IDENTIFIED BY password
WITH BACKUP;
```

10

General Considerations of Using Transparent Data Encryption

When you use Transparent Data Encryption, you should consider factors such as security, performance, and storage overheads.

10.1 Migrating Encrypted TDE Columns or Tablespaces after a Database Upgrade from Release 11g

If you are upgrading from Oracle Database release 11g, then you must manually convert the typed master encryption keys to united mode master encryption keys.

You must perform this task for master encryption keys in both column encryption and tablespace encryption configurations.

1. Navigate to directory that has the TDE password-based keystore `ewallet.p12`.
2. Check the keystore contents and look for typed keys (that is, identifiers starting with `ORACLE.SECURITY.TS.ENCRYPTION.B` as follows).

   ```
   mkstore -wrl ewallet.p12 -list
   Enter wallet password: wallet_password
   ```

   Output similar to the following appears. The entries with **TS** are the typed keys that must be converted.

   ```
   Oracle Secret Store entries:
   ORACLE.SECURITY.DB.ENCRYPTION.AYTGHzMNxU9kv77shWptsxkAAAAAAAAAAAAAAAAAAAAAA
   AAAAAAA
   ORACLE.SECURITY.DB.ENCRYPTION.ARjWMYe05E9Bv4KULq1no8UAAAAAAAAAAAAAAAAAAAAAA
   AAAAAAA
   ORACLE.SECURITY.DB.ENCRYPTION.ASoO2D5jtk/
   jv9FPW6rhhGcAAAAAAAAAAAAAAAAAAAAAAAAAAAA
   ORACLE.SECURITY.DB.ENCRYPTION.MASTERKEY
   ORACLE.SECURITY.TS.ENCRYPTION.BW8QuLUTgOpV0PtEt/
   lR028CAwAAAAAAAAAAAAAAAAAAAAAAAAAA
   ORACLE.SECURITY.TS.ENCRYPTION.BWNFXbAc38AZMdFdi2kuWV0CAwAAAAAAAAAAAAAAAAAA
   AAAAAAA
   ```

3. Back up the keystore.

   For example:

   ```
   cp ewallet.p12 ewallet_ddddmmyy.p12
   ```

4. Create a united key for each typed key that you found in the keystore.

   To accomplish this, you must modify the identifier name (Key ID) to have **DB** instead of **TS** and the first letter from **B** to **A**. For this you must add a new entry to the keystore. The

following example shows how to convert the first typed key in the list that was shown in the preceding steps.

a. List the value for identifier to double-check it. The bold text indicates the two changes to be made.

   For example:

   ```
   ORACLE.SECURITY.TS.ENCRYPTION.BW8QuLUTgOpV0PtEt/
   lR028CAwAAAAAAAAAAAAAAAAA
   ```

   This example must be changed as follows:

   ```
   ORACLE.SECURITY.DB.ENCRYPTION.AW8QuLUTgOpV0PtEt/
   lR028CAwAAAAAAAAAAAAAAAAA
   ```

b. Using this updated entry, add a new entry to the keystore.

   For example:

   ```
   mkstore -wrl wallet_location -viewEntry
   ORACLE.SECURITY.TS.ENCRYPTION.BW8QuLUTgOpV0PtEt/
   lR028CAwAAAAAAAAAAAAAAAAAAAAAAAAA
   ```

   The `-viewEntry` option of this command displays the identifier value for the TS encryption, which you must include in the next command when you create the keystore entry:

   ```
   mkstore -wrl wallet_location -
   createEntry ORACLE.SECURITY.DB.ENCRYPTION.AW8QuLUTgOpV0PtEt/
   lR028CAwAAAAAAAAAAAAAAAA
   AAAAAAAAAA value_for_ts_encryption_identifier
   ```

c. For an auto-login keystore, rename the old auto-login keystore and then create a new one.

   i. Back up the master encryption key. For example:

      ```
      mv cwallet.sso cwallet_ddddmmyy.sso
      ```

   ii. Use the `ADMINISTER KEY MANAGEMENT` command to create the auto-login keystore.

Related Topics

- Creating an Auto-Login or a Local Auto-Login TDE Wallet
  As an alternative to password-protected TDE wallets, you can create either an auto-login or local auto-login TDE wallet.

10.2 Compression and Data Deduplication of Encrypted Data

When you use Oracle's compression products and options with Transparent Data Encryption (TDE) tablespace encryption, Oracle Database automatically applies compression before performing the encryption.

The compression products include the following: Oracle Recovery Manager (Oracle RMAN) compression, Oracle Advanced Compression, Exadata Hybrid Columnar Compression (EHCC), and so on.

This ensures that you receive the maximum space and performance benefits from compression, while also receiving the security of encryption at rest. In the `CREATE TABLESPACE` SQL statement, include both the `COMPRESS` and `ENCRYPT` clauses.

With column encryption, Oracle Database compresses the data after it encrypts the column. This means that compression will have minimal effectiveness on encrypted columns. There is one notable exception: if the column is a SecureFiles LOB, and the encryption is implemented with SecureFiles LOB Encryption, and the compression (and possibly deduplication) are implemented with SecureFiles LOB Compression & Deduplication, then compression is performed before encryption. Similar to the `CREATE TABLESPACE` statement for tablespace encryption, include both the `COMPRESS` and `ENCRYPT` clauses.

> **Note:**
>
> The type of compression that you can use depends on the licensing that you have for Advanced Compression. See *Oracle Database Licensing Information User Manual*.

Related Topics

- *Oracle Database Backup and Recovery User's Guide*

10.3 Security Considerations for Transparent Data Encryption

As with all Oracle Database features, you should consider security when you create TDE policies.

10.3.1 Transparent Data Encryption General Security Advice

Security considerations for Transparent Data Encryption (TDE) operate within the broader area of total system security.

Follow these general guidelines:

- Identify the degrees of sensitivity of data in your database, the protection that they need, and the levels of risk to be addressed. For example, highly sensitive data requiring stronger protection can be encrypted with the AES256 algorithm. A database that is not as sensitive can be protected with no salt or the `nomac` option to enable performance benefits.
- Evaluate the costs and benefits that are acceptable to data and keystore protection. Protection of keys determines the type of keystore to be used: auto-login TDE wallets, password-based TDE wallets, or external keystores.
- Consider having separate security administrators for TDE and for the database.
- Consider having a separate and exclusive keystore for TDE.
- Implement protected back-up procedures for your encrypted data.

10.3.2 Transparent Data Encryption Column Encryption-Specific Advice

Additional security considerations apply to normal database and network operations when using TDE.

Encrypted column data stays encrypted in the data files, undo logs, redo logs, and the buffer cache of the system global area (SGA). However, data is decrypted during expression

evaluation, making it possible for decrypted data to appear in the swap file on the disk. Privileged operating system users can potentially view this data.

Column values encrypted using TDE are stored in the data files in encrypted form. However, these data files may still contain some plaintext fragments, called ghost copies, left over by past data operations on the table. This is similar to finding data on the disk after a file was deleted by the operating system.

10.3.3 Managing Security for Plaintext Fragments

You should remove old plaintext fragments that can appear over time.

Old plaintext fragments may be present for some time until the database overwrites the blocks containing such values. If privileged operating system users bypass the access controls of the database, then they might be able to directly access these values in the data file holding the tablespace.

To minimize this risk:

1. Create a new tablespace in a new data file.

   You can use the `CREATE TABLESPACE` statement to create this tablespace.

2. Move the table containing encrypted columns to the new tablespace. You can use the `ALTER TABLE.....MOVE` statement.

   Repeat this step for all of the objects in the original tablespace.

3. Drop the original tablespace.

   You can use the `DROP TABLESPACE` *`tablespace`* `INCLUDING CONTENTS KEEP DATAFILES` statement. Oracle recommends that you securely delete data files using platform-specific utilities.

4. Use platform-specific and file system-specific utilities to securely delete the old data file. Examples of such utilities include `shred` (on Linux) and `sdelete` (on Windows).

10.4 Performance and Storage Overhead of Transparent Data Encryption

The performance of Transparent Data Encryption can vary.

10.4.1 Performance Overhead of Transparent Data Encryption

Transparent Data Encryption tablespace encryption has small associated performance overhead.

The actual performance impact on applications can vary. TDE column encryption affects performance only when data is retrieved from or inserted into an encrypted column. No reduction in performance occurs for operations involving unencrypted columns, even if these columns are in a table containing encrypted columns. Accessing data in encrypted columns involves small performance overhead, and the exact overhead you observe can vary.

The total performance overhead depends on the number of encrypted columns and their frequency of access. The columns most appropriate for encryption are those containing the most sensitive data.

Enabling encryption on an existing table results in a full table update like any other `ALTER TABLE` operation that modifies table characteristics. Keep in mind the potential performance and redo log impact on the database server before enabling encryption on a large existing table.

A table can temporarily become inaccessible for write operations while encryption is being enabled, TDE table keys are being rekeyed, or the encryption algorithm is being changed. You can use online table redefinition to ensure that the table is available for write operations during such procedures.

If you enable TDE column encryption on a very large table, then you may need to increase the redo log size to accommodate the operation.

Encrypting an indexed column takes more time than encrypting a column without indexes. If you must encrypt a column that has an index built on it, you can try dropping the index, encrypting the column with `NO SALT`, and then re-creating the index.

If you index an encrypted column, then the index is created on the encrypted values. When you query for a value in the encrypted column, Oracle Database transparently encrypts the value used in the SQL query. It then performs an index lookup using the encrypted value.

> **Note:**
>
> If you must perform range scans over indexed, encrypted columns, then use TDE tablespace encryption in place of TDE column encryption.

Related Topics

- Creating an Encrypted Column in an External Table
  The external table feature enables you to access data in external sources as if the data were in a database table.
- *Oracle Database Administrator's Guide*

10.4.2 Storage Overhead of Transparent Data Encryption

TDE tablespace encryption has no storage overhead, but TDE column encryption has some associated storage overhead.

Encrypted column data must have more storage space than plaintext data. In addition, TDE pads out encrypted values to multiples of 16 bytes. This means that if a credit card number requires nine bytes for storage, then an encrypted credit card value will require an additional seven bytes.

Each encrypted value is also associated with a 20-byte integrity check. This does not apply if you have encrypted columns using the `NOMAC` parameter. If data was encrypted with salt, then each encrypted value requires an additional 16 bytes of storage.

The maximum storage overhead for each encrypted value is from one to 52 bytes.

Related Topics

- Creating an Encrypted Column in an External Table
  The external table feature enables you to access data in external sources as if the data were in a database table.

10.5 Modifying Your Applications for Use with Transparent Data Encryption

You can modify your applications to use Transparent Data Encryption.

1. Configure the software or external keystore for TDE, and then set the master encryption key.
2. Verify that the master encryption key was created by querying the `KEY_ID` column of the `V$ENCRYPTION_KEYS` view.
3. Identify the sensitive columns (such as those containing credit card data) that require Transparent Data Encryption protection.
4. Decide whether to use TDE column encryption or TDE tablespace encryption.

   See the following sections for more information:

   - How Transparent Data Encryption Column Encryption Works
   - How Transparent Data Encryption Tablespace Encryption Works
5. Open the keystore.
6. Encrypt the columns or tablespaces.

   See the following sections for more information:

   - Encrypting Columns in Tables
   - Encryption Conversions for Tablespaces and Databases

10.6 How ALTER SYSTEM and orapki Map to ADMINISTER KEY MANAGEMENT

Many of the clauses from the `ALTER SYSTEM` statement correspond to the `ADMINISTER KEY MANAGEMENT` statement.

Table 10-1 compares the Transparent Data Encryption usage of the `ALTER SYSTEM` statement and the `orapki` utility from previous releases with the `ADMINISTER KEY MANAGEMENT` statement.

Table 10-1 How ALTER SYSTEM and orapki Map to ADMINISTER KEY MANAGEMENT

Behavior	ALTER SYSTEM or orapki	ADMINISTER KEY MANAGEMENT
Creating a keystore	For TDE wallets: `ALTER SYSTEM SET ENCRYPTION KEY ["certificate_ID"] IDENTIFIED BY keystore_password;` For external keystores, the keystore is available after you configure the external key manager.	For TDE wallets: `ADMINISTER KEY MANAGEMENT CREATE KEYSTORE IDENTIFIED BY TDE_wallet_password` For external keystores, the keystore is available after you configure the external key manager.

Table 10-1 (Cont.) How ALTER SYSTEM and orapki Map to ADMINISTER KEY MANAGEMENT

Behavior	ALTER SYSTEM or orapki	ADMINISTER KEY MANAGEMENT
Creating an auto-login keystore	`orapki wallet create -wallet wallet_location -auto_login [-pwd password]`	For TDE wallets: `ADMINISTER KEY MANAGEMENT CREATE [LOCAL] AUTO_LOGIN KEYSTORE FROM KEYSTORE IDENTIFIED BY TDE_wallet_password;` This type of keystore applies to TDE wallets only.
Opening a keystore	`ALTER SYSTEM SET [ENCRYPTION] WALLET OPEN IDENTIFIED BY keystore_password;`	`ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN [FORCE KEYSTORE] IDENTIFIED BY keystore_password | EXTERNAL STORE [CONTAINER = ALL | CURRENT];`
Closing a keystore	`ALTER SYSTEM SET [ENCRYPTION] WALLET CLOSE IDENTIFIED BY keystore_password;`	For both TDE wallets and external keystores: `ADMINISTER KEY MANAGEMENT SET KEYSTORE CLOSE [IDENTIFIED BY keystore_password] [CONTAINER = ALL | CURRENT];`
Reverse migrating from an external key manager to a TDE wallet	`ALTER SYSTEM SET [ENCRYPTION] KEY IDENTIFIED BY wallet_password REVERSE MIGRATE USING "external_key_manager_password";`	`ADMINISTER KEY MANAGEMENT SET [ENCRYPTION] KEY IDENTIFIED BY keystore_password REVERSE MIGRATE USING "external_key_manager_password" WITH BACKUP [USING 'backup_identifier'];`
Migrating from a TDE wallet to Oracle Key Vault or Oracle Cloud Interface (OCI) Key Management Service (KMS)	`ALTER SYSTEM SET [ENCRYPTION] KEY IDENTIFIED BY "Oracle_Key_Vault_password" MIGRATE USING wallet_password;`	`ADMINISTER KEY MANAGEMENT SET [ENCRYPTION] KEY IDENTIFIED BY "Oracle_Key_Vault_password" MIGRATE USING wallet_password;`
Changing a keystore password	`orapki wallet change_pwd -wallet wallet_location [-oldpwd password ] [-newpwd password]`	For password-based TDE wallets: `ADMINISTER KEY MANAGEMENT ALTER KEYSTORE PASSWORD IDENTIFIED BY TDE_wallet_old_password SET TDE_wallet_new_password [WITH BACKUP [USING 'backup_identifier']];` For external keystores, you close the keystore, change it in the external key manager interface, and then reopen the keystore.
Backing up a password-based TDE wallet	No `ALTER SYSTEM` or `orapki` equivalent for this functionality	`ADMINISTER KEY MANAGEMENT BACKUP KEYSTORE [USING 'backup_identifier'] IDENTIFIED BY TDE_wallet_password [TO 'TDE_wallet_location'];`

Table 10-1 (Cont.) How ALTER SYSTEM and orapki Map to ADMINISTER KEY MANAGEMENT

Behavior	ALTER SYSTEM or orapki	ADMINISTER KEY MANAGEMENT
Merging two TDE wallets into a third new keystore	No `ALTER SYSTEM` or `orapki` equivalent for this functionality	`ADMINISTER KEY MANAGEMENT MERGE KEYSTORE 'TDE_wallet1_location' [IDENTIFIED BY TDE_wallet1_password] AND KEYSTORE 'TDE_wallet2_location' [IDENTIFIED BY TDE_wallet2_password] INTO NEW KEYSTORE 'TDE_wallet3_location' IDENTIFIED BY TDE_wallet3_password;`
Merging one TDE wallet into another existing keystore	No `ALTER SYSTEM` or `orapki` equivalent for this functionality	`ADMINISTER KEY MANAGEMENT MERGE KEYSTORE 'TDE_wallet1_location' [IDENTIFIED BY TDE_wallet1_password] INTO EXISTNG KEYSTORE 'TDE_wallet2_location' IDENTIFIED BY TDE_wallet2_password [WITH BACKUP [USING 'backup_identifier']];`
Setting or rekeying the master encryption key	`ALTER SYSTEM SET [ENCRYPTION] KEY IDENTIFIED BY keystore_password;` **Note:** The `ALTER SYSTEM SET ENCRYPTION KEY` statement does not update the `V$ENCRYPTION_KEYS` dynamic view after you rekey the encryption key.	`ADMINISTER KEY MANAGEMENT SET ENCRYPTION KEY [USING TAG 'tag'] IDENTIFIED BY keystore_password WITH BACKUP [USING 'backup_identifier'] [CONTAINER = ALL | CURRENT];` After you rekey the encryption key, the `V$ENCRYPTION_KEYS` dynamic view is updated.
Creating a master encryption key for later user	No `ALTER SYSTEM` or `orapki` equivalent for this functionality	`ADMINISTER KEY MANAGEMENT CREATE KEY [USING TAG 'tag'] IDENTIFIED BY keystore_password [WITH BACKUP [USING 'backup_identifier']] [CONTAINER = (ALL|CURRENT)];`
Activating a master encryption key	No `ALTER SYSTEM` or `orapki` equivalent for this functionality	`ADMINISTER KEY MANAGEMENT USE KEY 'key_identifier' [USING TAG 'tag'] IDENTIFIED BY keystore_password [WITH BACKUP [USING 'backup_identifier']];`
Creating custom tags for master encryption keys	No `ALTER SYSTEM` or `orapki` equivalent for this functionality	`ADMINISTER KEY MANAGEMENT SET TAG 'tag' FOR 'master_key_identifier' IDENTIFIED BY keystore_password [WITH BACKUP [USING 'backup_identifier']];`
Exporting a master encryption key	No `ALTER SYSTEM` or `orapki` equivalent for this functionality	`ADMINISTER KEY MANAGEMENT EXPORT [ENCRYPTION] KEYS WITH SECRET "export_secret" TO 'file_path' IDENTIFIED BY TDE_wallet_password [WITH IDENTIFIER IN 'key_id1', 'key_id2', 'key_idn' | (SQL_query)]`

Table 10-1 (Cont.) How ALTER SYSTEM and orapki Map to ADMINISTER KEY MANAGEMENT

Behavior	ALTER SYSTEM or orapki	ADMINISTER KEY MANAGEMENT
Importing a master encryption key	No `ALTER SYSTEM` or `orapki` equivalent for this functionality	`ADMINISTER KEY MANAGEMENT IMPORT [ENCRYPTION] KEYS WITH SECRET "import_secret" | FROM 'file_name' IDENTIFIED BY TDE_wallet_password [WITH BACKUP [USING 'backup_identifier']];`

Storing Oracle Database secrets in a keystore	No `ALTER SYSTEM` or `orapki` equivalent for this functionality	For TDE wallets:
		`ADMINISTER KEY MANAGEMENT ADD SECRET|UPDATE SECRET|DELETE SECRET "secret" FOR CLIENT 'client_identifier' [USING TAG'tag'] IDENTIFIED BY TDE_wallet_password [WITH BACKUP [USING 'backup_identifier'];`
		For external keystores:
		`ADMINISTER KEY MANAGEMENT ADD SECRET|UPDATE SECRET|DELETE SECRET "secret" FOR CLIENT 'client_identifier' [USING TAG 'tag'] IDENTIFIED BY "external_key_manager_password";`

10.7 Data Loads from External Files to Tables with Encrypted Columns

You can use SQL*Loader to perform data loads from files to tables that have encrypted columns.

Be aware that with SQL*Loader, you cannot include the `ENCRYPT` clause in the column definition of an external table of the type `ORACLE_LOADER`, but you can include it in the column definitions of external tables of type `ORACLE_DATAPUMP`.

- External tables of type `ORACLE_LOADER`

  The reason that you cannot include the `ENCRYPT` clause in the column definitions of external tables of the type `ORACLE_LOADER` is because the contents of an external table with the `ORACLE_LOADER` type must come from a user-specified plaintext "backing file," and such plaintext files cannot contain any TDE encrypted data.

  If you use the `ENCRYPT` clause in the definition of an external table of type `ORACLE_LOADER`, then when you query the TDE-encrypted column in this external table, the query fails. This is because TDE expects the external data to have been encrypted, and automatically tries to decrypt it on load. This action fails because the "backing file" only contains plaintext.

- External tables of type `ORACLE_DATAPUMP`

  You can use TDE column encryption with external tables of type `ORACLE_DATAPUMP`. This is because for external tables of `ORACLE_DATAPUMP` type, the "backing file" is always created by Oracle Database(during an unload operation) and thus does have support for being populated with encrypted data.

10.8 Transparent Data Encryption and Database Close Operations

You should ensure that the software or external keystore is open before you close the database.

The master encryption keys may be required during the database close operation. The database close operation automatically closes the software or external keystore.

Related Topics

- Step 2: Open the TDE Wallet
  Depending on the type of TDE wallet you create, you must manually open the wallet before you can use it.
- Step 2: Open the Connection to Oracle Key Vault
  After you have configured the database to use Oracle Key Vault for TDE key management, you must open the connection to Oracle Key Vault before you can use it.

11

Using Transparent Data Encryption with Other Oracle Features

You can use Oracle Data Encryption with other Oracle features, such as Oracle Data Guard or Oracle Real Application Clusters.

11.1 How Transparent Data Encryption Works with Export and Import Operations

Oracle Data Pump can export and import tables that contain encrypted columns, as well as encrypt entire dump sets.

11.1.1 About Exporting and Importing Encrypted Data

You can use Oracle Data Pump to export and import tables that have encrypted columns.

For both software and external keystores, the following points are important when you must export tables containing encrypted columns:

- Sensitive data should remain unintelligible during transport.
- Authorized users should be able to decrypt the data after it is imported at the destination.

When you use Oracle Data Pump to export and import tables containing encrypted columns, it uses the `ENCRYPTION` parameter to enable encryption of data in dump file sets. The `ENCRYPTION` parameter allows the following values:

- `ENCRYPTED_COLUMNS_ONLY`: Writes encrypted columns to the dump file set in encrypted format
- `DATA_ONLY`: Writes all of the data to the dump file set in encrypted format
- `METADATA_ONLY`: Writes all of the metadata to the dump file set in encrypted format
- `ALL`: Writes all of the data and metadata to the dump file set in encrypted format
- `NONE`: Does not use encryption for dump file sets

11.1.2 Exporting and Importing Tables with Encrypted Columns

You can export and import tables with encrypted columns using the `ENCRYPTION=ENCRYPTED_COLUMNS_ONLY` setting.

1. Ensure that the keystore is open before you attempt to export tables containing encrypted columns.

   If you are exporting data in a pluggable database (PDB), then ensure that the wallet is open in the PDB. If you are exporting into the root, then ensure that the wallet is open in the root.

   To find if the keystore is open, query the `STATUS` column of the `V$ENCRYPTION_WALLET` view. If you must open the keystore, then run the following SQL statement:

```
ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN
IDENTIFIED BY TDE_wallet_password
[CONTAINER = ALL | CURRENT];
```

The *TDE_wallet_password* setting is the password for the keystore. The keystore must be open because the encrypted columns must be decrypted using the TDE table keys, which requires access to the TDE master encryption key. The columns are reencrypted using a password, before they are exported.

2. Run the `EXPDP` command, using the `ENCRYPTION_PASSWORD` parameter to specify a password that is used to encrypt column data in the export dump file set.

   The following example exports the `employee_data` table. The `ENCRYPTION_PWD_PROMPT = YES` setting enables you to prompt for the password interactively, which is a recommended security practice.

   ```
   expdp hr TABLES=employee_data DIRECTORY=dpump_dir
   DUMPFILE=dpcd2be1.dmp ENCRYPTION=ENCRYPTED_COLUMNS_ONLY
   ENCRYPTION_PWD_PROMPT = YES

   Password: password_for_hr
   ```

3. To import the exported data into the target database, ensure that you specify the same password that you used for the export operation, as set by the `ENCRYPTION_PASSWORD` parameter.

   The password is used to decrypt the data. Data is reencrypted with the new TDE table keys generated in the target database. The target database must have the keystore open to access the TDE master encryption key. The following example imports the `employee_data` table:

   ```
   impdp hr TABLES=employee_data DIRECTORY=dpump_dir
   DUMPFILE=dpcd2be1.dmp
   ENCRYPTION_PWD_PROMPT = YES

   Password: password_for_hr
   ```

11.1.3 Using Oracle Data Pump to Encrypt Entire Dump Sets

Oracle Data Pump can encrypt entire dump sets, not just Transparent Data Encryption columns.

While importing, you can use either the password or the keystore TDE master encryption key to decrypt the data. If the password is not supplied, then the TDE master encryption key in the keystore is used to decrypt the data. The keystore must be present and open at the target database. The open keystore is also required to reencrypt column encryption data at the target database.

You can use the `ENCRYPTION_MODE=TRANSPARENT` setting to transparently encrypt the dump file set with the TDE master encryption key stored in the keystore. A password is not required in this case. The keystore must be present and open at the target database, and it must *contain* the TDE master encryption key from the *source* database for a successful decryption of column encryption metadata during an import operation.

The open keystore is also required to reencrypt column encryption metadata at the target database. If a keystore already exists on the target database, then you can export the current TDE master encryption key *from* the keystore of the source database and import it *into* the keystore of the target database.

- Use the `ENCRYPTION_MODE` parameter to specify the encryption mode. `ENCRYPTION_MODE=DUAL` encrypts the dump set using the TDE master encryption key stored in the keystore and the password provided.

For example, to use dual encryption mode to export encrypted data:

```
expdp hr DIRECTORY=dpump_dir1
DUMPFILE=hr_enc.dmp
ENCRYPTION=all
ENCRYPTION_PASSWORD=encryption_password
ENCRYPTION_PWD_PROMPT=yes
ENCRYPTION_ALGORITHM=AES256
ENCRYPTION_MODE=dual

Password: password_for_hr
Encryption Password: password_for_encryption
```

Related Topics

- Exporting and Importing the TDE Master Encryption Key
  You can export and import the TDE master encryption key in different ways.
- *Oracle Database Utilities*
- Creating an Encrypted Column in an External Table
  The external table feature enables you to access data in external sources as if the data were in a database table.

11.1.4 Using Oracle Data Pump with Encrypted Data Dictionary Data

Oracle Data Pump operations provide protections for encrypted passwords and other encrypted data.

When you enable the encryption of fixed-user database passwords in a source database, then an Oracle Data Pump export operation dump stores a known invalid password for the database link password. This password is in place instead of the encrypted password that the export operation extracts from the database. An `ORA-39395: Warning: object <database link name> requires password reset after import` warning message is displayed as a result. If you import data into an Oracle Database 18*c* or later database, then this same warning appears when the database link object with its invalid password is created in the target database. When this happens, you must reset the database link password, as follows:

```
ALTER DATABASE LINK database_link_name CONNECT TO schema_name IDENTIFIED BY password;
```

To find information about the database link, you can query the `V$DBLINK` dynamic view.

When the encryption of fixed-user database passwords has been disabled in a source database, then there are no changes to Data Pump. The obfuscated database link passwords are exported and imported as in previous releases.

In this case, Oracle recommends the following:

- Set the `ENCRYPTION_PASSWORD` parameter on the `expdp` command so that you can further protect the obfuscated database link passwords.
- Set the `ENCRYPTION_PWD_PROMPT` parameter to `YES` so that the password can be entered interactively from a prompt, instead of being echoed on the screen.

Both the `ENCRYPTION_PASSWORD` and the `ENCRYPTION_PWD_PROMPT` parameters are available in import operations. `ENCRYPTION_PWD_PROMPT` is only available with the `expdp` and `impdp`

command-line clients, whereas `ENCRYPTION_PASSWORD` is available in both the command-line clients and the `DBMS_DATAPUMP` PL/SQL package.

During an import operation, whether the keystore is open or closed affects the behavior of whether or not an encryption password must be provided. If the keystore was open during the export operation and you provided an encryption password, then you do not need to provide the password during the import operation. If you use `AUTOLOGIN`, then the wallet will be open by default. If the keystore is closed during the export operation, then you must provide the password during the import operation.

Related Topics

- *Oracle Database Reference*
- *Oracle Database Utilities*

11.2 How Transparent Data Encryption Works with Oracle Data Guard

In Oracle Data Guard, primary and standby databases need to have access to the same encryption.

Either the TDE wallet needs to be copied over to the standby databases, or all primary and standby databases endpoints need to have access to the same virtual wallet or encryption key in Oracle Key Vault. For more information, see Configuring Wallet-Based Transparent Data Encryption in Oracle Data Guard and Configuring TDE and Oracle Key Vault in an Oracle Data Guard Environment.

11.2.1 About Using Transparent Data Encryption with Oracle Data Guard

For both TDE wallets and external keystores, Oracle Data Guard supports Transparent Data Encryption (TDE).

If the primary database uses TDE, then each standby database in a Data Guard configuration must have a copy of the encryption keystore from the primary database. If you reset the TDE master encryption key in the primary database, then you must copy the keystore from the primary database that contains the TDE master encryption key to each standby database.

Note the following:

- Re-key operations with wallet-based TDE will cause the Managed Recovery Process (MRP) on the standby databases to fail because the new TDE master encryption key is not yet available. In order to circumvent this problem, you can split the `set key` command into the `create key` and `use key` commands. After each command, copy the wallets to the standby database and 'force open' the keystore.
- Encrypted data in log files remains encrypted when data is transferred to the standby database. Encrypted data also stays encrypted during transit.

Related Topics

- Merging TDE Wallets
  You can merge TDE wallets in a variety of ways.
- Configuring Wallet-Based Transparent Data Encryption in Oracle Data Guard
  You can configure wallet-based Transparent Data Encryption (TDE) in an Oracle Data Guard environment.

11.2.2 Encryption of Tablespaces in an Oracle Data Guard Environment

You can control tablespace encryption in the primary and standby databases in an Oracle Data Guard environment.

11.2.2.1 About the Encryption of Tablespaces in an Oracle Data Guard Environment

In an Oracle Data Guard environment, you can control the automatic encryption of tablespaces in both the primary and standby databases, for on-premises and Oracle Cloud Infrastructure (OCI) environments.

To control the encryption of new tablespaces, you set the `TABLESPACE_ENCRYPTION` initialization parameter.

Oracle recommends that you encrypt primary databases. However, because encryption requirements may vary depending on the site, you can use the `TABLESPACE_ENCRYPTION` parameter to configure a mixed encryption environment for on-premises and in-Cloud environments.

Note the following about using the `TABLESPACE_ENCRYPTION` parameter:

- Redo decryption takes place at the redo transport level.
- If you do not have an Advanced Security Option (ASO) license, which includes Transparent Data Encryption, then you can configure an un-encrypted on-premises primary database to have an encrypted standby database in Oracle Cloud Infrastructure (OCI). Even after a role transition (planned or unplanned), the new on-premises standby database remains un-encrypted, while the primary database in OCI is encrypted. See the YouTube video Hybrid Oracle Data Guard without Transparent Data Encryption (TDE) License.
- If the `ENCRYPT_NEW_TABLESPACES` setting that you choose conflicts with the `TABLESPACE_ENCRYPTION` setting, then `TABLESPACE_ENCRYPTION` takes precedence.

  > **Note:**
  >
  > In Oracle Database releases 19.16 and 23ai, the `ENCRYPT_NEW_TABLESPACES` was deprecated, to be replaced with `TABLESPACE_ENCRYPTION`.

- You must set `TABLESPACE_ENCRYPTION` in the CDB root, not in any PDBs.
- The default `TABLESPACE_ENCRYPTION` setting for OCI databases is `AUTO_ENABLE`. The setting is mandatory, and any changes to it are ignored.
- The default `TABLESPACE_ENCRYPTION` setting for on-premises databases is `MANUAL_ENABLE`.

In an Oracle Data Guard environment that uses on-premises databases and Oracle Base Database Service or Oracle Exadata Cloud (ExaCS), you can configure tablespace encryption in either of the following scenarios:

- **Encrypt the tablespace in the Cloud standby database but not in the on-premises primary database:** When an unencrypted on-premises primary database creates an unencrypted tablespace, adds a data file, or updates a table, then the redo is not encrypted. However, when the encrypted Cloud standby database applies the redo, it ensures that the tablespace or data file is created, or that the block is updated, and are all encrypted in the Cloud. After the switchover operation, if the encrypted Cloud primary

database adds an implicitly encrypted tablespace or data file, or updates a table, then the tablespace is encrypted. The unencrypted on-premises standby database must decrypt the redo, create an unencrypted tablespace or data file, and then ensure that the block is not encrypted on-premises.

- **Encrypt the tablespace in the Cloud primary database but not in the on-premises standby database:** When an encrypted Cloud primary database creates an implicitly encrypted tablespace, adds a data file, or updates a table, then the redo is also encrypted. The unencrypted on-premises standby database must decrypt the redo and ensure that the tablespace and data file are created, or the updated files are all unencrypted. After the switchover operation, if the unencrypted on-premises primary database adds an unencrypted tablespace or data file, or updates a table, then the redo is not encrypted. The encrypted Cloud standby database adds an encrypted tablespace or data file, and then ensures that the block is encrypted in the Cloud.

For example, if you want to use `TABLESPACE_ENCRYPTION` in a configuration that followed the best practice of having both on-premises and OCI databases encrypted, then you would set `TABLESPACE_ENCRYPTION` to `AUTO_ENABLE` for both the on-premises and OCI databases. Alternatively, if the on-premises database is not encrypted in a hybrid disaster recovery configuration with Oracle Base Database Service or Oracle ExaCS, for example, you could set `TABLESPACE_ENCRYPTION` to `DECRYPT_ONLY`. The OCI database is set to `AUTO_ENABLE` by default.

See also the video Hybrid Oracle Data Guard without Transparent Data Encryption (TDE) License.

11.2.2.2 Configuring the Encryption of Tablespaces in an Oracle Data Guard Environment

To configure the hybrid encryption of tablespaces, you must set the `TABLESPACE_ENCRYPTION` initialization parameter.

1. Log in to the CDB root of the primary or the standby database instance by using SQL*Plus with the `SYSDBA` administrative privilege.

   You cannot set the `TABLESPACE_ENCRYPTION` parameter in a pluggable database (PDB).

2. Set the `TABLESPACE_ENCRYPTION` initialization parameter as follows:

   ```
   ALTER SYSTEM SET TABLESPACE_ENCRYPTION = 'value' SCOPE = SPFILE SID = '*';
   ```

   In this specification, replace *`value`* with one of the following settings:

   - `AUTO_ENABLE` encrypts all new tablespaces if the database is licensed for Oracle Advanced Security. This is the default setting for Cloud databases.
     Note the following:
     - If an existing tablespace is not encrypted, then the database writes a warning to the alert log.
     - Encrypted tablespaces cannot be converted to unencrypted tablespaces in:
       * Oracle Exadata Database Service on Cloud@Customer
       * Oracle Autonomous Database on Exadata Cloud@Customer and
       * all cloud databases including
         * Oracle Base Database Service
         * Oracle Exadata Database Service on Dedicated Infrastructure

        - Oracle Exadata Database Service on Exascale Infrastructure
        - Oracle Autonomous Database Serverless
        - Oracle Autonomous Database on Dedicated Exadata Infrastructure
      - Because all tablespaces must be encrypted in the Cloud, setting this parameter to `DECRYPT_ONLY` or `MANUAL_ENABLE` on a Cloud database will result in an error message.
      - In the primary database, this setting automatically encrypts new tablespaces.
      - The standby databases follow the primary database and also automatically encrypt new tablespaces.
  - `DECRYPT_ONLY` prevents new tablespaces from being encrypted. Use this setting if you do not have a TDE license for your on-premises primary database that you want to protect with an encrypted standby database in OCI. See the Hybrid Oracle Data Guard without Transparent Data Encryption (TDE) License video. This setting is designed for sites that do not have the Advanced Security Option.
  - `MANUAL_ENABLE` enables you to selectively encrypt tablespaces if the database is licensed for Oracle Advanced Security. This is the default for both on-premises primary and standby databases and it uses the same behavior as in previous Oracle Database releases.

  In an Oracle Real Application Clusters (Oracle RAC) environment, set `TABLESPACE_ENCRYPTION` to the same value for all instances of the primary database, and for all instances of the standby database. Because the default value is `MANUAL_ENABLE`, Oracle recommends that during an upgrade to the current release of Oracle Database, until all database instances are rolled over and upgraded, to not change `TABLESPACE_ENCRYPTION` for any of these database instances. When all the database instances are upgraded, then you can modify the `TABLESPACE_ENCRYPTION` parameter.

3. Set the master encryption key on the primary database for the root, if you have not done so already.
4. Set master encryption keys on all the PDBs associated with this root, if you have not done so already.
5. Copy the wallet from the primary database to the standby database.

Note the following with regard to rekey operations:

- Modifying the `TABLESPACE_ENCRYPTION` parameter does not affect master key rotation operations.
  If a tablespace key rotation is triggered on the primary database, then the standby database will attempt to rotate the key for the tablespace as well. However if the standby tablespace is unencrypted and does not have a key, then it will generate an error because there is no key to regenerate. If the standby tablespace is unencrypted but it has inherited a key from primary because of the `DECRYPT_ONLY` setting, then the key will be rotated. In either case, it does not affect the unencrypted tablespace.
- Both the master encryption key and the tablespace key rotation can only be performed on the primary database.
- When a tablespace key rotation is performed on the primary database, then the standby database will attempt to rotate the key for the tablespace as well. However if the standby tablespace is unencrypted, then the rekey operation attempt will generate an error, because there is no key to regenerate.

Related Topics

- Offline Encryption of Existing Tablespaces
  You can perform offline encryption by using the `ALTER TABLESPACE` or `ALTER DATABASE DATAFILE SQL` statements with the `ENCRYPT` or `DECRYPT` clauses.
- Configuring a TDE Wallet and TDE Master Encryption Key for United Mode
  In united mode, the TDE wallet resides in the CDB root but the master keys from this wallet are available for the PDBs that have their TDE wallets in united mode.
- Configuring a TDE Wallet and TDE Master Encryption Key in Isolated Mode
  In isolated mode, the TDE wallet is associated with a PDB.

11.2.2.3 Encrypting an Existing Tablespace in Oracle Data Guard with Online Conversion

To encrypt an existing tablespace in an Oracle Data Guard environment with online conversion, use `ALTER TABLESPACE` with the `ONLINE` and `ENCRYPT` clauses.

1. Connect to the CDB root or PDB as a user who has been granted the `ALTER TABLESPACE` privilege.
2. Ensure that the `COMPATIBLE` initialization parameter is set to 12.2.0.0 or later.

   You can use the `SHOW PARAMETER` command to check the current setting of a parameter.
3. Ensure that the database is open in read-write mode.

   You can query the `STATUS` column of the `V$INSTANCE` dynamic view to find if a database is open and the `OPEN_MODE` column of the `V$DATABASE` view to find if it in read-write mode.
4. If necessary, open the primary database in read-write mode.

   ```
   ALTER DATABASE OPEN READ WRITE;
   ```
5. Ensure that the auxiliary space is at least the same size as the largest data file of this tablespace.

   This size requirement is because Oracle Database performs the conversion one file at a time. For example, if the largest data file of the tablespace is 32 GB, then ensure that you have 32 GB of auxiliary space. To find the space used by a data file, query the `BYTES` or `BLOCKS` column of the `V$DATAFILE` dynamic performance view.
6. Optionally, in the CDB root of the standby database, set the `DB_RECOVERY_AUTO_REKEY` parameter to `OFF` to prevent the standby from falling behind due to the additional computing resources that are needed to encrypt the standby databases. (Use this step if the standby databases are not powerful enough to handle the additional load of encryption.)

   This setting prevents the standby recovery from performing an automatic rekey operation on every data file, but it will remember the new key that the primary database used.

   For example:

   ```
   ALTER SYSTEM SET DB_RECOVERY_AUTO_REKEY = OFF SCOPE = BOTH;
   ```
7. As a user who has been granted the `ADMINISTER KEY MANAGEMENT` or `SYSKM` privilege, create a TDE wallet, open the wallet, and set the first TDE master encryption key.

   For example:

   ```
   ADMINISTER KEY MANAGEMENT CREATE KEYSTORE 'TDE_wallet_location' IDENTIFIED BY
   TDE_wallet_password;
   ```

```
ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN IDENTIFIED BY TDE_wallet_password;
ADMINISTER KEY MANAGEMENT SET KEY IDENTIFIED BY TDE_wallet_password WITH BACKUP;
```

8. Run the `ALTER TABLESPACE` statement using the `ENCRYPTION` and `ENCRYPT` clauses to perform the encryption.

   For example, for a non-Oracle managed files tablespace named `users`:

   ```
   ALTER TABLESPACE USERS ENCRYPTION ONLINE ENCRYPT FILE_NAME_CONVERT = ('users.dbf',
   'users_enc.dbf');
   ```

   In this example:

   - `ENCRYPTION ONLINE ENCRYPT` sets the statement to encrypt the tablespace `USERS` while it is online and encrypts with the AES256 encryption algorithm (and XTS cipher mode) by default. For the `SYSTEM` tablespace, you can use the `ENCRYPT` clause to encrypt the tablespace, but you cannot specify an encryption algorithm because it is encrypted with the default algorithm (AES256 with XTS in Oracle Database 23ai). After encrypting the `SYSTEM` tablespace, use the `REKEY` clause to specify the algorithm.
   - `FILE_NAME_CONVERT` specifies one or more pairs of data files that are associated with the tablespace. The first name in the pair is an existing data file, and the second name is for the encrypted version of this data file, which will be created after the `ALTER TABLESPACE` statement successfully runs. If the tablespace has more than one data file, then you must process them all in this statement. Note the following:
     - Separate each file name with a comma, including multiple pairs of files. For example:

       ```
       FILE_NAME_CONVERT = ('users1.dbf', 'users1_enc.dbf', 'users2.dbf',
       'users2_enc.dbf')
       ```

     - You can specify directory paths in the `FILE_NAME_CONVERT` clause. For example, the following clause converts and moves the matching files of the tablespace from the `dbs` directory to the `dbs/enc` directory:

       ```
       FILE_NAME_CONVERT = ('dbs', 'dbs/enc')
       ```

     - The `FILE_NAME_CONVERT` clause recognizes patterns. The following example converts the data files `users_1.dbf` and `users_2.dbf` to `users_enc1.dbf` and `users_enc2.dbf`:

       ```
       FILE_NAME_CONVERT = ('users', 'users_enc')
       ```

     - In an Oracle Data Guard environment, include the name of the standby database data file in the `FILE_NAME_CONVERT` settings.
     - If you are using Oracle-managed file mode, then the new file name is internally assigned, so this file name should not affect your site's file-naming standards. If you are using non-Oracle-managed file mode and if you omit the `FILE_NAME_CONVERT` clause, then Oracle Database internally assigns an auxiliary file name, and then later renames it back to the original name. This enables the encryption process to use the name that you had originally given the file to be encrypted. The renaming operation is effectively creating another copy of the file, hence it is slower than explicitly including the `FILE_NAME_CONVERT` clause. For better performance, include the `FILE_NAME_CONVERT` clause.
     - You can find the data files for a tablespace by querying the `V$DATAFILE` or `V$DATAFILE_HEADER` dynamic views.

   By default, data files are in the `$ORACLE_HOME/dbs` directory. If the data files are located there, then you do not have to specify a path.

9. Monitor the standby's `STATUS` column in the `V$ENCRYPTED_TABLESPACE` dynamic view.
10. If you had set the `DB_RECOVERY_AUTO_REKEY to OFF`, when `V$ENCRYPTED_TABLESPACE` in the standby database shows `ENCRYPTING` (or `REKEYING` if the tablespace is already encrypted), run the one of the following statements on the standby:
    - Run `ALTER TABLESPACE <TBS_NAME> ENCRYPTION FINISH ENCRYPT` to encrypt the tablespaces of the standby database.
    - Run `ALTER TABLESPACE <TBS_NAME> ENCRYPTION FINISH REKEY` to rekey the already encrypted tablespaces in the standby database.

After you complete the conversion, you can check the encryption status by querying the `STATUS` column of the `V$ENCRYPTED_TABLESPACES` dynamic view. The `ENCRYPTIONALG` column of this view shows the encryption algorithm that is used. If the conversion process was interrupted, then you can resume it by running `ALTER TABLESPACE` with the `FINISH` clause. For example, if the primary data file converts but the standby data file does not, then you can run `ALTER TABLESPACE ... FINISH` on the standby database for the standby data files.

Related Topics

- Best Practice after DBCA Creates an Encrypted Database
  After DBCA has created an encrypted stand-alone or Oracle Data Guard primary and standby database, you can implement Transparent Data Encryption (TDE) best practices.
- Setting the COMPATIBLE Initialization Parameter for Tablespace Encryption
  To set the `COMPATIBLE` initialization parameter, you must edit the initialization parameter file for the database instance.
- Finishing an Interrupted Online Encryption Conversion
  If an online encryption process is interrupted, then you can complete the conversion by rerunning the `ALTER TABLESPACE` statement using the `FINISH` clause.

11.2.3 Configuring TDE and Oracle Key Vault in an Oracle Data Guard Environment

You can configure Oracle Data Guard in a multitenant environment so that it can work with TDE and Oracle Key Vault.

The following scenario shows the configuration with Oracle Key Vault in a single-instance, multitenant Oracle Data Guard environment with one physical standby database. The version for the primary and standby databases must be release 23.6 or later. To complete this procedure, you must perform each step in the sequence shown. After you complete the procedure, Oracle Data Guard will use Oracle Key Vault for TDE key management exclusively, and there will be no TDE wallet on your database servers. Oracle recommends that you monitor the alert logs of both primary and standby databases.

1. Download the Oracle Key Vault deployment script that the Oracle Key Vault administrators prepared to enable database administrators to automatically register their Oracle databases with Oracle Key Vault.

   Oracle Key Vault RESTful Services Administrator's Guide has an example of how to create a script to automatically enroll Oracle databases as endpoints. The deployment scripts reside on a shared file system from which database administrators can download. There are two different versions of these deployment scripts. The `primary.zip` file is for the primary database, and the `secondary.zip` file is for all standby databases. You can use these scripts for an Oracle Data Guard or an Oracle RAC environment.

Another component that the Oracle Key Vault administrators prepare and add to the deployment script is a configuration file that contains all details for the deployment scripts to connect to Oracle Key Vault.

2. Copy the two deployment scripts (`primary.zip` and `secondary.zip`) that an Oracle Key Vault administrator created for database administrators to download from a shared location.

   a. Copy the `primary.zip` file to the primary database.

   ```
   $ scp user@ip_address:/path/to/file/primary.zip .
   ```

   b. Copy the `secondary.zip` file to the standby database.

   ```
   $ scp user@ip_address:/path/to/file/secondary.zip .
   ```

3. On their respective servers, extract the zip files.

   ```
   $ unzip primary.zip

   $ unzip secondary.zip
   ```

4. Run the `primary-run-me.sh` and `secondary-run-me.sh` scripts, which contain the commands for the RESTful API to run in Oracle Key Vault.

   The Oracle Key Vault RESTful services will run these commands in order to register this database in Oracle Key Vault with unique wallet and endpoint names.

   a. **Primary database:** For example:

   ```
   $ more primary-run-me.sh

   #!/bin/bash
   export EP_NAME=${ORACLE_SID^^}_on_${hostname -s}
   export WALLET_NAME=${ORACLE_SID^^}
   curl -Ok --tlsv1.2 https://Oracle_Key_Vault_IP_address:5695/okvrestclipackage.zip
   unzip -oj okvrestclipackage.zip lib/okvrestcli.jar -d ./lib
   cat > /home/oracle/deploy-primary.sh  << EOF
   #!/bin/bash
   mkdir -pv<WALLET_ROOT>/okv
   okv manage-access wallet create --wallet ${WALLET_NAME} --unique FALSE
   okv admin endpoint create --endpoint ${EP_NAME} --description "$(hostname -f), $
   (hostname -i)"
       --type ORACLE_DB --platform LINUX64 --subgroup "USE CREATOR SUBGROUP" --
   unique FALSE --strict-ip-check TRUE
   okv manage-access wallet set-default --wallet ${WALLET_NAME} --endpoint $
   {EP_NAME}
   expect << _EOF
       set timeout 120
       spawn okv admin endpoint provision --endpoint ${EP_NAME} --location
   <WALLET_ROOT>/okv --auto-login FALSE
       expect "Enter Oracle Key Vault endpoint password: "
       send "change-on-install\r"
       expect eof
   _EOF
   EOF
   ```

   b. **Standby database:** For example:

   ```
   $ more secondary-run-me.sh

   #!/bin/bash
   export EP_NAME=${ORACLE_SID^^}_on_$(HOSTNAME/.*)
   ```

```
export WALLET_NAME=${ORACLE_SID^^}
curl -Ok --tlsv1.2 https://Oracle_Key_Vault_IP_address:5695/okvrestclipackage.zip
unzip -oj okvrestclipackage.zip lib/okvrestcli.jar -d ./lib
cat > /home/oracle/deploy-standby.sh << EOF
#!/bin/bash
okv admin endpoint create --endpoint ${EP_NAME} --description "$(hostname -f) $
(hostname -i)" --subgroup "USE CREATOR SUBGROUP" --unique FALSE
okv admin endpoint update --endpoint ${EP_NAME} --strict-ip-check TRUE
okv manage-access wallet set-default --wallet ${WALLET_NAME} --endpoint $
{EP_NAME}
expect << _EOF
    set timeout 120
    spawn okv admin endpoint provision --endpoint ${EP_NAME} --location
<WALLET_ROOT>/okv --auto-login FALSE
    expect "Enter Oracle Key Vault endpoint password: "
    send "change-on-install\r"
    expect eof
_EOF
EOF
```

5. Create the following directories on the primary database and the standby database.

   For example:

   ```
   $ mkdir -pv /u01/opt/oracle/product/okv
   $ mkdir -pv /u01/opt/oracle/product/tde
   $ mkdir -pv /u01/opt/oracle/product/tde_seps
   ```

   In this specification:

   - The `/u01/opt/oracle/product` directory will be defined as `WALLET_ROOT` in a later step.
   - `/u01/opt/oracle/product/okv` is the installation directory for the Oracle Key Vault client software. Depending on how the `TDE_CONFIGURATION` parameter is set, the Oracle Database will look for the Oracle Key Vault client software in *`wallet_root/okv`*.
   - `/u01/opt/oracle/product/tde` will store an auto-login wallet, which only contains the future Oracle Key Vault password, enabling an auto-login Oracle Key Vault configuration. Depending on how `TDE_CONFIGURATION` is set, the Oracle Database will look for the TDE wallet or an auto-open wallet for Oracle Key Vault, in *`wallet_root/tde`*.
   - `/u01/opt/oracle/product/tde_seps` will store an auto-login wallet, which only contains the future Oracle Key Vault password. This will hide the Oracle Key Vault password from the SQL*Plus command line and potentially from the database administrator to enforce separation of duties between Oracle database administrators and Oracle Key Vault administrators.

6. Run the RESTful API on the primary database first, because the deployment script on the standby databases depends on the presence of the shared virtual wallet in Oracle Key Vault that the script on the primary database creates.

   - Primary database:

     ```
     $ ./deploy-primary.sh
     ```

   - Standby databases:

     ```
     $ ./deploy-standby.sh
     ```

7. Perform the following steps on the primary and the standby.

   a. On the primary and standby databases, run the `root.sh` script to deploy the PKCS#11 library.

```
# /u01/opt/oracle/product/okv/bin/root.sh
```

The following output should appear:

```
Creating directory: /opt/oracle/extapi/64/hsm/oracle/1.0.0/
Copying PKCS library to /opt/oracle/extapi/64/hsm/oracle/1.0.0/
Setting PKCS library file permissions
```

b. As `root`, or with `sudo` privileges, create the following directories

These directories must be owned by root, have 755 as their file and directory permissions, and there must not be softlinks. The Oracle Key Vault versions are fictitious and are used to explain the functionality.

```
$ sudo sh -c 'mkdir -pvm755 /opt/oracle/extapi/64/pkcs11/okv/lib/
{21.6,21.7}'
```

c. Move the PKCS#11 library from the legacy directory in to the new directory that you just created that matches you current Oracle Key Vault version (for example 21.6):

```
$ sudo sh -c 'mv -v /opt/oracle/extapi/64/hsm/oracle/1.0.0/
liborapkcs.so /opt/oracle/extapi/64/pkcs11/okv/lib/21.6/'
```

d. Confirm the new tree structure.

```
$ tree -n /opt/oracle/extapi/64
```

e. As a user with the `ALTER SYSTEM` privilege, run the following statement on the primary and standby databases:

```
ALTER SYSTEM SET PKCS11_LIBRARY_LOCATION = '/opt/oracle/extapi/64/
pkcs11/okv/lib/21.6/liborapkcs.so' SCOPE = SPFILE;
```

f. Shut down the primary, shut down the standby, then start the standby and the primary (in this order) to apply the configuration changes.

g. At this stage, from now on, after upgrading the Oracle Key Vault server and client, your database does not need to restart to load the updated PKCS#11 library. Instead, after the updated PKCS#11 library has been moved into `/opt/oracle/extapi/64/pkcs11/okv/lib/21.7/liborapkcs.so`, execute as a user with the `SYSKM` privilege:

```
ADMINISTER KEY MANAGEMENT
SWITCHOVER TO LIBRARY '/opt/oracle/extapi/64/pkcs11/okv/lib/21.7/
liborapkcs.so'
FOR ALL CONTAINERS;
```

8. Run the `okvutil changepwd` command to change the password for the wallet that you installed, starting from the primary database and then to the standby.

Because all database administrators downloaded the same deployment script, all databases have the same password into Oracle Key Vault. This step enables each database to have a unique password.

```
$ /u01/opt/oracle/product/okv/bin/okvutil changepwd -t wallet -l /u01/opt/oracle/
product/okv/ssl/

Enter wallet password: default_password
Enter new wallet password: Oracle_Key_Vault_password
```

```
Confirm new wallet password: Oracle_Key_Vault_password
Wallet password changed successfully
```

9. On the primary and standby databases, run the following statements.

   a. Run the following statement to add the Oracle Key Vault password as a secret into an auto-open wallet to replace the Oracle Key Vault password in the SQL*Plus command line with `EXTERNAL STORE`.

   ```
   ADMINISTER KEY MANAGEMENT ADD SECRET 'Oracle_Key_Vault_password'
   FOR CLIENT 'OKV_PASSWORD'
   TO LOCAL AUTO_LOGIN KEYSTORE '/u01/opt/oracle/product/tde_seps';
   ```

   b. Run the following statement to add the Oracle Key Vault password as a secret into an auto-open wallet to enable auto-open Oracle Key Vault.

   ```
   ADMINISTER KEY MANAGEMENT ADD SECRET 'Oracle_Key_Vault_password'
   FOR CLIENT 'OKV_PASSWORD'
   TO LOCAL AUTO_LOGIN KEYSTORE '/u01/opt/oracle/product/tde';
   ```

   c. Configure the primary and standby databases to always encrypt new tablespaces:

   ```
   ALTER SYSTEM SET TABLESPACE_ENCRYPTION = 'value' SCOPE = SPFILE SID = '*';
   ```

   d. In the primary and standby databases, define the `WALLET_ROOT` static initialization parameter:

   ```
   ALTER SYSTEM SET WALLET_ROOT = '/u01/opt/oracle/product' SCOPE = SPFILE;
   ```

   e. Restart the primary and standby databases so that the preceding `ALTER SYSTEM SET WALLET_ROOT` and `ALTER SYSTEM SET TABLESPACE_ENCRYPTION` statements take effect.

   f. After the database restarts, configure TDE to use Oracle Key Vault as the first keystore and the auto-open wallet in `WALLET_ROOT/tde` as the secondary keystore.

   Run the following statement in both the primary and standby databases:

   ```
   ALTER SYSTEM SET TDE_CONFIGURATION = "KEYSTORE_CONFIGURATION=OKV|FILE" SCOPE =
   BOTH;
   ```

10. In the primary database, create your first TDE master encryption keys in Oracle Key Vault.

    Check the `alert.log` of the standby database. The managed recovery process (MRP) should not be stopped, since the standby database finds the correct master key in the shared virtual wallet in Oracle Key Vault.

    a. **Primary root container:** Set the first master encryption key.

    For all `ADMINISTER KEY MANAGEMENT` statements that do not change the TDE configuration, the password will be replaced by `EXTERNAL STORE`. This enables separation of duties between the database administrators and the Oracle Key Vault administrators because the Oracle Key Vault administrators do not need to share the Oracle Key Vault password with the database administrators.

    ```
    sqlplus sys as syskm
    Enter password: password

    ADMINISTER KEY MANAGEMENT SET KEY FORCE KEYSTORE IDENTIFIED BY EXTERNAL STORE
    CONTAINER = CURRENT;
    ```

    b. **All primary PDBs:** Set the first, tagged, master key for each open PDB. The benefit of tagging the PDB keys is that they can later be easily identified to belong to a certain PDB.

    ```
    SELECT ' ADMINISTER KEY MANAGEMENT SET KEY
    USING TAG '''||UPPER(SYS_CONTEXT('USERENV', 'CON_NAME'))||' '||TO_CHAR
    (SYS_EXTRACT_UTC (SYSTIMESTAMP),
    ```

```
'YYYY-MM-DD HH24:MI:SS"Z"')||''' FORCE KEYSTORE IDENTIFIED BY EXTERNAL STORE;'
AS "SET KEY COMMAND";
```

c. Run the generated output of this `SELECT` statement.

For example:

```
ADMINISTER KEY MANAGEMENT SET KEY
USING TAG 'pdb_name date time'
FORECE KEYSTORE IDENTIFIED BY EXTERNAL STORE;
```

11. Perform the following steps in the root container.

a. Encrypt the `USERS`, `SYSTEM`, and `SYSAUX` tablespaces in the root container. Encrypting `TEMP` and `UNDO` is optional, because data from encrypted tablespaces is tracked and written to `TEMP` and `UNDO` in encrypted form.

For example:

```
ALTER TABLESPACE USERS ENCRYPTION ONLINE;
ALTER TABLESPACE SYSTEM ENCRYPTION ONLINE ENCRYPT;
ALTER TABLESPACE SYSAUX ENCRYPTION ONLINE ENCRYPT;
```

b. Observe the `alert.log` of the standby database to confirm that the standby tablespaces are also encrypted.

12. Encrypt the PDB tablespaces:

a. Encrypt the `USERS`, `SYSTEM`, `SYSAUX` and all application tablespaces in the PDBs.

Encrypting the `TEMP` and `UNDO` tablespaces is optional because data from encrypted tablespaces is tracked and automatically encrypted before being written into `TEMP` or `UNDO`.

```
ALTER TABLESPACE USERS ENCRYPTION ONLINE [USING 'algorithm'] ENCRYPT;
ALTER TABLESPACE SYSTEM ENCRYPTION ONLINE ENCRYPT;
ALTER TABLESPACE SYSAUX ENCRYPTION ONLINE [USING 'algorithm'] ENCRYPT;
```

The `SYSTEM` tablespace can only be encrypted with the database default algorithm, which is `AES256` unless it has been changed in step 9. If you want to encrypt the `SYSTEM` tablespace with another algorithm, then you can rekey the `SYSTEM` tablespace, for example: For example:

```
ALTER TABLESPACE SYSTEM ENCRYPTION ONLINE USING 'any_supported_algorithm' REKEY;
```

Observe the `alert.log` of the standby database to confirm the encryption and rekey operations are applied there as well.

b. Optionally, encrypt the `UNDO` and `TEMP` tablespaces.

For example, to encrypt an `UNDO` tablespace named `UNDOTBS1`:

```
ALTER TABLESPACE UNDOTBS1 ENCRYPTION ONLINE [USING 'algorithm'] ENCRYPT;
```

You cannot use the `ALTER TABLESPACE` statement to encrypt an existing `TEMP` tablespac. To encrypt a `TEMP` tablespace, you must create a new one and encrypt it. First, extract the DDL that was used to create the original `TEMP` tablespace.

```
SELECT DBMS_METADATA.GET_DDL ('TABLESPACE', 'TEMP') FROM DBA_TEMP_FILES;
```

Modify the output so that tempfile and tablespace name are different. For example:

```
CREATE TEMPORARY TABLESPACE "TEMP_ENC"
TEMPFILE '/u01/opt/oracle/oradata/${ORACLE_SID}/${PDB-NAME}/
temp01_enc.dbf'
SIZE 146800640 AUTOEXTEND ON NEXT 655360
MAXSIZE 32767M EXTENT MANAGEMENT LOCAL UNIFORM SIZE 1048576;
```

You only need to add the `ENCRYPTION [USING 'algorithm'] ENCRYPT` clause if you need to apply an algorithm other than the default. Omitting the `ENCRYPTION [USING 'algorithm'] ENCRYPT` clause will automatically encrypt the `TEMP` tablespace with the default algorithm (AES256 with XTS cipher code).

Next, make this new encrypted `TEMP` tablespace the default temp tablespace of this PDB, and then drop the old clear-text `TEMP` tablespace.

```
ALTER DATABASE DEFAULT TEMPORARY TABLESPACE TEMP_ENC;
DROP TABLESPACE TEMP INCLUDING CONTENTS AND DATAFILES;
```

13. Optionally, create a tablespace and table in the primary database PDB.

    When you create the tablespace in the primary database without encryption keywords in that statement. It will be encrypted with `AES256` by default unless the database default algorithm has been changed in an earlier step, when you changed the database default algorithm from `AES256` to either `AES128` or `AES192`.

    ```
    CREATE TABLESPACE protected DATAFILE SIZE 50M;

    CREATE TABLE SYSTEM.TEST TABLESPACE protected
    AS SELECT * FROM DBA_OBJECTS;
    ```

14. Confirm that you can select from the table that is stored in an encrypted tablespace.

    ```
    SELECT COUNT(*), OWNER FROM SYSTEM.TEST
    GROUP BY OWNER
    ORDER BY 1 DESC;
    ```

15. On the **primary** and **standby** databases, run the following statements in the root:

    ```
    select distinct c.name AS PDB_NAME, t.name
    AS TBS_NAME, nvl(e.encryptionalg, '----')
    AS ENC_ALG, nvl(e.ciphermode, '---')
    AS "MODE", nvl(e.status, '----')
    AS ENC_STATUS from v$containers c, v$tablespace t, v$encrypted_tablespaces e
    WHERE c.con_id != 2 and e.ts#(+) = t.ts# and c.con_id(+) = t.con_id order by 1, 3
    desc, 2;

    PDB_NAME        TBS_NAME        ENC_ALG MODE  ENC_STATUS
    --------------- --------------- ------- ----- ----------
    CDB$ROOT        SYSAUX          AES256  XTS   NORMAL
    CDB$ROOT        SYSTEM          AES256  XTS   NORMAL
    CDB$ROOT        USERS           AES256  XTS   NORMAL
    CDB$ROOT        TEMP            ----    ---   ----
    CDB$ROOT        UNDOTBS1        ----    ---   ----
    FINPDB23        PROTECTED       AES256  XTS   NORMAL
    FINPDB23        SYSAUX          AES256  XTS   NORMAL
    FINPDB23        SYSTEM          AES256  XTS   NORMAL
    FINPDB23        USERS           AES256  XTS   NORMAL
    FINPDB23        TEMP            ----    ---   ----
    FINPDB23        UNDOTBS1        ----    ---   ----
    ```

Note that the `TEMP` tablespace is not listed in the output of the standby databases.

16. Optionally, validate the configuration.

    a. Perform an Oracle Data Guard switchover between the primary and standby databases.

       See *Oracle Data Guard Concepts and Administration*.

       Perform the following steps in the new primary database.

    b. Select from the encrypted table in your PDB.

       Because there is an auto-open connection into Oracle Key Vault, the following query does not require that you enter the Oracle Key Vault password.

```
SELECT COUNT(*), OWNER FROM SYSTEM.TEST
GROUP BY OWNER
ORDER BY 1 DESC;

24 rows selected.
```

    c. Rekey the PDB.

```
SELECT ' ADMINISTER KEY MANAGEMENT SET KEY
USING TAG '''||UPPER(SYS_CONTEXT('USERENV', 'CON_NAME'))||' '||TO_CHAR
(SYS_EXTRACT_UTC (SYSTIMESTAMP),
'YYYY-MM-DD HH24:MI:SS"Z"')||'''
FORCE KEYSTORE
IDENTIFIED BY EXTERNAL STORE;'
AS "RE-KEY COMMAND" FROM DUAL;
```

    d. Run the generated output of this `SELECT` statement.

       For example:

```
ADMINISTER KEY MANAGEMENT SET KEY
USING TAG 'pdb_name date time'
IDENTIFIED BY EXTERNAL STORE;
```

    e. Perform another Oracle Data Guard switchover.

       See *Oracle Data Guard Concepts and Administration*.

    f. Select from the encrypted table in your PDB.

       Because there is an auto-open connection into Oracle Key Vault, the following query does not require that you enter the Oracle Key Vault password.

```
SELECT COUNT(*), OWNER FROM SYSTEM.TEST
GROUP BY OWNER
ORDER BY 1 DESC;

24 rows selected.
```

    g. Rekey the PDB.

```
SELECT ' ADMINISTER KEY MANAGEMENT SET KEY
USING TAG '''||UPPER(SYS_CONTEXT('USERENV', 'CON_NAME'))||' '||TO_CHAR
(SYS_EXTRACT_UTC (SYSTIMESTAMP),
'YYYY-MM-DD HH24:MI:SS"Z"')||'''
FORCE KEYSTORE
IDENTIFIED BY EXTERNAL STORE;'
AS "RE-KEY COMMAND" FROM DUAL;
```

    h. Run the generated output of this s`SELECT`tatement.

       For example:

```
ADMINISTER KEY MANAGEMENT SET KEY
USING TAG 'pdb_name date time'
IDENTIFIED BY EXTERNAL STORE;
```

Related Topics

- Setting the Tablespace Encryption Default Algorithm
  The `TABLESPACE_ENCRYPTION_DEFAULT_ALGORITHM` applies to specific encryption scenarios.

11.2.4 Configuring Wallet-Based Transparent Data Encryption in Oracle Data Guard

You can configure wallet-based Transparent Data Encryption (TDE) in an Oracle Data Guard environment.

The following scenario shows how to configure TDE wallet-based TDE in a single-instance, multitenant Oracle Data Guard environment with one physical standby database. The version for the primary and standby databases must be release 23.6 or later. To complete this procedure, you must perform each step in the sequence shown. After you complete this configuration, you can migrate the primary and standby databases from the TDE wallet to Oracle Key Vault. Oracle recommends that you monitor the alert logs of both primary and standby databases.

1. In the primary and standby databases, create the directories that are needed for a TDE wallet-based TDE configuration.

   For example:

   ```
   # mkdir -pv /etc/ORACLE/KEYSTORES/finance/tde_seps
   # cd /etc
   # chown -Rv oracle:oinstall ./ORACLE
   # chmod -Rv 700 ./ORACLE
   ```

2. In the CDB root of the primary and standby databases, as a user who has the `ALTER SYSTEM` privilege, run the following statements in SQL*Plus to set the appropriate parameters.

   ```
   ALTER SYSTEM SET WALLET_ROOT = '/etc/ORACLE/KEYSTORES/${ORACLE_SID}' SCOPE
   = SPFILE;
   ALTER SYSTEM SET TABLESPACE_ENCRYPTION = 'AUTO_ENABLE' SCOPE = SPFILE;
   ```

3. Restart the primary and standby databases.
4. Set the following parameters in the primary and standby databases:

   ```
   ALTER SYSTEM SET TDE_CONFIGURATION = "KEYSTORE_CONFIGURATION=FILE" SCOPE =
   BOTH;
   ```

5. Configure tablespace encryption.
   a. On the primary database, as a user with the `SYSKM` privilege, create a password-protected and a (local) auto-open TDE wallet.

      When you create this keystore, Oracle Database automatically creates the *`WALLET_ROOT`*`/tde` directory.

      ```
      ADMINISTER KEY MANAGEMENT CREATE KEYSTORE IDENTIFIED BY
      TDE_wallet_password;
      ```

```
ADMINISTER KEY MANAGEMENT CREATE LOCAL AUTO_LOGIN KEYSTORE FROM
KEYSTORE IDENTIFIED BY TDE_wallet_password;
```

b. Enable separation of duties by hiding the TDE wallet password in another (local) auto-open wallet to replace the keystore password with `EXTERNAL STORE` for those `ADMINISTER KEY MANAGEMENT` commands that do not change the TDE configuration in the primary and all standby databases:

```
ADMINISTER KEY MANAGEMENT ADD SECRET 'TDE_wallet_password'
FOR CLIENT 'TDE_WALLET'
TO LOCAL AUTO_LOGIN KEYSTORE '/etc/ORACLE/KEYSTORES/finance/tde_seps';
```

6. Set the first key in the `CDB$ROOT` container of the primary database.

```
ADMINISTER KEY MANAGEMENT SET KEY FORCE KEYSTORE IDENTIFIED BY EXTERNAL
STORE WITH BACKUP CONTAINER = CURRENT;
```

7. Log in to each open primary PDB and set the first, tagged key.

```
SELECT 'ADMINISTER KEY MANAGEMENT SET KEY USING TAG
'''||UPPER(SYS_CONTEXT('USERENV', 'CON_NAME'))||' '||TO_CHAR
(SYS_EXTRACT_UTC (SYSTIMESTAMP), 'YYYY-MM-DD HH24:MI:SS"Z"')||'''
FORCE KEYSTORE IDENTIFIED BY EXTERNAL STORE
WITH BACKUP;' AS "SET KEY COMMAND";
```

8. Copy the TDE wallets from primary to standby databases.

   The local auto-open TDE wallet of the primary cannot be opened on the standby, so it is excluded from the copy process. For example:

```
$ rsync -rvpt --exclude '*.sso' /etc/ORACLE/KEYSTORES/finance/tde/
standby_host:/etc/ORACLE/KEYSTORES/finance/tde/
```

9. On the CDB root of the standby, as a user who has the `SYSKM` administrative privilege, create a local auto-login TDE wallet:

```
ADMINISTER KEY MANAGEMENT CREATE LOCAL AUTO_LOGIN KEYSTORE
FROM KEYSTORE IDENTIFIED BY TDE_wallet_password;
```

10. On the CDB root of the primary, encrypt the `users` tablespace.

```
ALTER TABLESPACE USERS ENCRYPTION ONLINE ENCRYPT;
ALTER TABLESPACE SYSTEM ENCRYPTION ONLINE ENCRYPT;
ALTER TABLESPACE SYSAUX ENCRYPTION ONLINE ENCRYPT;
```

11. As a user with the `ALTER TABLESPACE` privilege, encrypt the `SYSTEM`, `SYSAUX`, `USERS`, and all sensitive application tablespaces in the PDB.

```
ALTER TABLESPACE USERS ENCRYPTION ONLINE ENCRYPT;
ALTER TABLESPACE SYSTEM ENCRYPTION ONLINE ENCRYPT;
ALTER TABLESPACE SYSAUX ENCRYPTION ONLINE ENCRYPT;
```

12. Optionally, encrypt the `UNDO` and `TEMP` tablespaces in the primary PDBs.

a. Perform the following `ALTER TABLESPACE` statement:

```
ALTER TABLESPACE UNDOTBS1 ENCRYPTION ONLINE MODE 'XTS' ENCRYPT;
```

b. Create a new encrypted `TEMP` tablespace with the same parameters that were applied to the original `TEMP` tablespace.

You cannot use the `ALTER TABLESPACE` statement to encrypt an existing `TEMP` tablespace, so you must create a new one.
First, extract the DDL that was used to create the original `TEMP` tablespace.

```
SELECT DBMS_METADATA.GET_DDL ('TABLESPACE', 'TEMP') FROM DBA_TEMP_FILES;
```

Second, modify the output so that the tempfile and tablespace names are different. For example:

```
CREATE TEMPORARY TABLESPACE "TEMP_ENC"
TEMPFILE '/u01/opt/oracle/oradata/${ORACLE_SID}/${PDB-NAME}/
temp01_enc.dbf'
SIZE 146800640 AUTOEXTEND ON NEXT 655360
MAXSIZE 32767M EXTENT MANAGEMENT LOCAL UNIFORM SIZE 1048576;
```

c. Make the new encrypted `TEMP` tablespace the default `TEMP` tablespace:

```
ALTER DATABASE DEFAULT TEMPORARY TABLESPACE TEMP_ENC;
```

d. Drop the old clear-text `TEMP` tablespace:

```
DROP TABLESPACE TEMP INCLUDING CONTENTS AND DATAFILES;
```

e. Rename the new `TEMP` tablesplace.

```
ALTER TABLESPACE TEMP_ENC RENAME TO TEMP;
```

13. Optionally, create a small sample tablespace (which will be encrypted with AES256 even if no `ENCRYPTION` syntax is given), and create a copy of the `DBA_OBJECTS` table in this encrypted sample tablespace.

```
CREATE TABLESPACE PROTECTED DATAFILE SIZE 50M;
CREATE TABLE SYSTEM.TEST TABLESPACE PROTECTED AS SELECT * FROM DBA_OBJECTS;
```

14. Optionally, select from the encrypted copy of the `DBA_OBJECTS` table.

For example:

```
SELECT OWNER, COUNT(*) FROM SYSTEM.test GROUP BY OWNER ORDER BY 2 DESC;

20 rows selected.
```

15. Connect to the CDB root of the primary and standby databases and confirm.

```
SELECT DISTINCT c.name AS PDB_NAME, t.name AS TBS_NAME,
nvl(e.encryptionalg, '----') AS ENC_ALG, nvl(e.ciphermode, '---') as
"MODE",
```

```
nvl(e.status, '----') AS ENC_STATUS FROM v$containers c, v$tablespace t,
v$encrypted_tablespaces e
WHERE c.con_id != 2 AND e.ts#(+) = t.ts# AND c.con_id(+) = t.con_id
ORDER BY 1, 3 desc, 2;
```

Output similar to the following should appear:

```
PDB_NAME        TBS_NAME        ENC_ALG MODE  ENC_STATUS
--------------- --------------- ------- ----- ----------
CDB$ROOT        SYSAUX          AES256  XTS   NORMAL
CDB$ROOT        SYSTEM          AES256  XTS   NORMAL
CDB$ROOT        USERS           AES256  XTS   NORMAL
CDB$ROOT        TEMP            ----    ---   ----
CDB$ROOT        UNDOTBS1        ----    ---   ----
FINPDB23        PROTECTED       AES256  XTS   NORMAL
FINPDB23        SYSAUX          AES256  XTS   NORMAL
FINPDB23        SYSTEM          AES256  XTS   NORMAL
FINPDB23        USERS           AES256  XTS   NORMAL
FINPDB23        TEMP            ----    ---   ----
FINPDB23        UNDOTBS1        ----    ---   ----
```

The standby does not have a `TEMP` tablespace, so it will not be listed here. The `PROTECTED` tablespace is encrypted because `TABLESPACE_ENCRYPTION` was set to `AUTO_ENABLE`; it also applies the database default cipher mode, `CFB`.

16. Connect to the `dgmgrl` utility on the standby as a user who has the `SYSDG` administrative privilege.
17. Perform a switchover operation.

    ```
    DGMGRL> switchover to 'standby_database';
    ```

18. To confirm the auto-open TDE wallet functionality, select from encrypted data (for example, the copy of the `DBA_OBJECTS` table in the PDB) of the new primary database after the role switch.

    For example:

    ```
    SELECT OWNER, COUNT(*) FROM SYSTEM.test GROUP BY OWNER ORDER BY 2 DESC;

    20 rows selected.
    ```

19. Copy the TDE wallet to the standby, excluding the local auto-open keystores, and automatically deleting the obsolete local auto-open keystore on the standby side.

    For example:

    ```
    $ rsync -rvpt --exclude '*.sso' --delete-excluded /etc/ORACLE/KEYSTORES/
    finance/tde/ SanDiego:/etc/ORACLE/KEYSTORES/finance/tde/
    ```

20. On the standby, re-create a local auto-open TDE wallet from the updated password-protected keystore so that the new key is immediately available to the standby database when needed.

```
ADMINISTER KEY MANAGEMENT CREATE LOCAL AUTO_LOGIN KEYSTORE
FROM KEYSTORE IDENTIFIED BY TDE_wallet_password;
```

21. Use the following `SELECT` statement to create an `ADMINISTER KEY MANAGEMENT USE KEY` command that inserts the correct *key-id*, and adds a tag to the key:

```
SELECT ' ADMINISTER KEY MANAGEMENT
USE KEY '''||KEY_ID||'''
USING TAG '''||SYS_CONTEXT('USERENV', 'CON_NAME')||' '||
TO_CHAR (SYS_EXTRACT_UTC (SYSTIMESTAMP), 'YYYY-MM-DD HH24:MI:SS"Z"')||'''
FORCE KEYSTORE IDENTIFIED BY EXTERNAL STORE
WITH BACKUP;' AS "USE KEY COMMAND"
FROM V$ENCRYPTION_KEYS
WHERE TAG IS NULL;
```

The `USE KEY` clause updates the associations in the TDE wallet, so the updated password-protected wallet is copied to the standby, again excluding the local auto-open TDE wallet and deleting the obsolete local auto-open keystore on the receiving standby side, for example, with the following command:

```
$ rsync -rvpt --exclude '*.sso' --delete-excluded /etc/ORACLE/KEYSTORES/
finance/tde/ SanDiego:/etc/ORACLE/KEYSTORES/finance/tde/
```

22. On the CDB root of the standby database, recreate the local auto-login TDE wallet.

```
ADMINISTER KEY MANAGEMENT CREATE LOCAL AUTO_LOGIN KEYSTORE FROM KEYSTORE
IDENTIFIED BY TDE_wallet_password;
```

23. Using the `dgmgrl` utility, perform a switchover operation to the standby.

    For example:

```
DGMGRL>  switchover to 'standby_database' wait 5;
```

24. Optionally, to confirm the auto-open functionality, select from an encrypted table (for example, the copy of the `DBA_OBJECTS` table).

```
SELECT OWNER, COUNT(*) FROM SYSTEM.test GROUP BY OWNER ORDER BY 2 DESC;

20 rows selected.
```

11.2.5 Migrating a TDE Wallet in an Oracle Data Guard Environment to Oracle Key Vault

After you have configured TDE wallet-based Transparent Data Encryption (TDE) in an Oracle Data Guard environment, you can migrate primary and standby databases to Oracle Key Vault, without downtime.

The following scenario shows how to configure one endpoint for each of the primary and standby databases, and then migrate the primary and standby databases from a TDE wallet to Oracle Key Vault. This scenario uses a single-instance, multitenant Oracle Data Guard environment with one physical standby database. The version for the primary and standby databases must be release 19.6 or later. To complete this procedure, you must perform each step in the sequence shown. See *Oracle Key Vault RESTful Services Administrator's Guide* for how an Oracle Key Vault administrator can create these files, which are later used by database administrators to automatically onboard their databases into Oracle Key Vault.

1. Copy the primary deployment script to the primary database host.

   For example:

   ```
   $ rsync -v 192.168.1.29:/directory_on_shared_server/OKVdeploy-DG-
   primary.tgz .
   ```

2. Copy the secondary deployment script to all standby database hosts.

   For example:

   ```
   $ rsync -v 192.168.1.29:/directory_on_shared_server/OKVdeploy-DG-
   standby.tgz .
   ```

3. On the primary and standby database hosts, extract the archive.

   ```
   $ tar -xzvf OKVdeploy-DG*
   ```

4. On the primary database, run the `primary-run-me.sh` script.

   This script creates the deployment script (`okv-ep.sh`), which contains unique names for the wallet and endpoint that it creates in Oracle Key Vault for the primary database.

   ```
   $ /primary-run-me.sh

     % Total    % Received % Xferd  Average Speed   Time    Time     Time
   Current
                                    Dload  Upload   Total   Spent    Left
   Speed
   100 3750k  100 3750k    0     0  13.5M      0 --:--:-- --:--:-- --:--:--
   13.5M
   Archive:  okvrestclipackage.zip
     inflating: ./lib/okvrestcli.jar
   #!/bin/bash
   mkdir -pv /etc/ORACLE/KEYSTORES/finance/okv
   okv manage-access wallet create --wallet FINANCE --unique FALSE
   okv admin endpoint create --endpoint FINANCE_on_SanDiego --description
   "SanDiego.us.oracle.com, 192.168.56.193"
     --subgroup "USE CREATOR SUBGROUP" --unique FALSE --strict-ip-check TRUE
   ```

```
okv manage-access wallet set-default --wallet FINANCE --endpoint
FINANCE_on_SanDiego
expect << _EOF
    set timeout 120
    spawn okv admin endpoint provision --endpoint FINANCE_on_SanDiego --
location /etc/ORACLE/KEYSTORES/finance/okv --auto-login FALSE
    expect "Enter Oracle Key Vault endpoint password: "
    send "change-on-install\r"
    expect eof
_EOF
```

5. On the standby databases, run the `secondary-run-me.sh` script.

   This script creates the deployment script (`okv-ep.sh`), which contains unique names for the endpoint that it creates in Oracle Key Vault for the standby databases. Note that the standby endpoint uses the primary wallet as its default wallet, so that primary and standby databases have access to same key material.

```
$ ./secondary-run-me.sh

  % Total    % Received % Xferd  Average Speed   Time    Time     Time
Current
                                 Dload  Upload   Total   Spent    Left
Speed
100 3750k  100 3750k    0     0  16.0M      0 --:--:-- --:--:-- --:--:--
16.1M
Archive:  okvrestclipackage.zip
  inflating: ./lib/okvrestcli.jar
#!/bin/bash
mkdir -pv /etc/ORACLE/KEYSTORES/finance/okv
okv admin endpoint create --endpoint FINANCE_on_Phoenix
  --description "Phoenix.us.oracle.com, 192.168.56.194"
  --subgroup "USE CREATOR SUBGROUP"
  --unique FALSE
  --strict-ip-check TRUE
okv manage-access wallet set-default --wallet FINANCE --endpoint
FINANCE_on_Phoenix
expect << _EOF
    set timeout 120
    spawn okv admin endpoint provision --endpoint FINANCE_on_Phoenix --
location /etc/ORACLE/KEYSTORES/finance/okv --auto-login FALSE
    expect "Enter Oracle Key Vault endpoint password: "
    send "change-on-install\r"
    expect eof
_EOF
```

6. On the primary database host, run the `okv-ep.sh` script.

   This script creates a wallet and endpoint for the primary database in Oracle Key Vault. It makes the wallet the default wallet of that endpoint. The following `okv admin endpoint provision` command downloads and installs the Oracle Key Vault endpoint software into the existing directory, *`WALLET_ROOT`*`/okv`.

```
$ ./deploy-OKV.sh

{
```

```
  "result" : "Success",
  "value" : {
    "status" : "PENDING",
    "locatorID" : "87E59AAD-0AAA-4AE8-ADCE-01D283ECE9C4"
  }
}
{
  "result" : "Success",
  "value" : {
    "status" : "PENDING",
    "locatorID" : "D13CC460-7BFB-451D-9998-DA387FC45783"
  }
}
{
  "result" : "Success"
}
spawn okv admin endpoint provision --endpoint finance_on_SanDiego --
location /etc/ORACLE/KEYSTORES/finance/okv --auto-login FALSE
Enter Oracle Key Vault endpoint password:
{
  "result" : "Success"
}
```

7. On the standby database hosts, run the `okv-ep.sh` script.

   This script creates an endpoint for the standby database in Oracle Key Vault. It makes the primary wallet the default wallet of that endpoint. The following `okv admin endpoint provision` command downloads and installs the Oracle Key Vault endpoint software into the existing directory *`WALLET_ROOT`*`/okv`.

```
$ ./deploy-OKV.sh

{
  "result" : "Success",
  "value" : {
    "status" : "PENDING",
    "locatorID" : "4208FBB5-5FD4-47EE-B3DB-FA8277BAFB84"
  }
}
{
  "result" : "Success"
}
spawn okv admin endpoint provision --endpoint finance_on_Phoenix --
location /etc/ORACLE/KEYSTORES/finance/okv --auto-login FALSE
Enter Oracle Key Vault endpoint password:
{
  "result" : "Success"
}
```

8. On the primary and standby database hosts, run the `root.sh` script in the *`WALLET_ROOT`*`/okv/bin`.

   This script deploys the PKCS#11 library into the correct destination directory.

```
$ sudo /etc/ORACLE/KEYSTORES/finance/okv/bin/root.sh
```

9. On the primary and standby databases, change the default password to a strong password that the database administrator should not know.

   This password is the same for all databases that used the deployment script.

```
$ /etc/ORACLE/KEYSTORES/finance/okv/bin/okvutil changepwd -l /etc/ORACLE/
KEYSTORES/finance/okv/ssl/ -t wallet

Enter wallet password: current_endpoint_password
Enter new wallet password: new_endpoint_password
Confirm new wallet password: new_endpoint_password
Wallet password changed successfully
```

10. Optionally, update the endpoint parameters.

    This step and the next two steps are optional. By default, the persistent cache is a password-protected wallet that is protected with the endpoint password from the preceding step. With the change in steps 10, 11, and 12, the persistent cache will be protected with a random password, which implies that the persistent cache will expire after database shutdown, and needs to be rebuilt after each database restart. Additionally, the persistent cache expiration time is changed to 4 hours, but the persistent cache refresh window is extended to 21 days.

```
$ okv admin endpoint update --generate-json-input | jq '.service.options
|= ({endpoint} | .endpointConfiguration
|= (.expirePkcs11PersistentCacheOnDatabaseShutdown = "TRUE"
| .pkcs11PersistentCacheRefreshWindow = "P21D"
| .pkcs11PersistentCacheTimeout = "PT4H"))'
> ./update-endpoint.json; more ./update-endpoint.json

{
  "service": {
    "category": "admin",
    "resource": "endpoint",
    "action": "update",
    "options": {
      "endpoint": "#VALUE",
      "endpointConfiguration": {
        "expirePkcs11PersistentCacheOnDatabaseShutdown": "TRUE",
        "pkcs11PersistentCacheRefreshWindow": "P21D",
        "pkcs11PersistentCacheTimeout": "PT4H"
      }
    }
  }
}
```

11. Optionally, on the primary, check the status of the endpoints on both the primary and standby databases.

    For example, for a primary with an endpoint called `finance_on_SanDiego` and a standby with an endpoint called `finance_on_Phoenix`:

```
$ okv admin endpoint check-status --endpoint finance_on_SanDiego
$ okv admin endpoint check-status --endpoint finance_on_Phoenix
```

12. After both endpoints are active, apply and re-use the `update-endpoint.json` file to update all endpoints of the primary and standby databases.:

```
$ okv admin endpoint update --from-json ./update-endpoint.json --endpoint
finance_on_SanDiego
$ okv admin endpoint update --from-json ./update-endpoint.json --endpoint
finance_on_Phoenix
```

13. On the primary database host, upload the current and retired master encryption keys from the TDE wallet into the virtual wallet in Oracle Key Vault.

    For example:

```
$ /etc/ORACLE/KEYSTORES/finance/okv/bin/okvutil upload -l /etc/ORACLE/
KEYSTORES/finance/tde/ -t wallet -g FINANCE -v 3
```

14. In the root of the primary, using SQL*Plus, add the Oracle Key Vault password from step 9 into the TDE wallet to enable an auto-open Oracle Key Vault connection.

```
ADMINISTER KEY MANAGEMENT ADD SECRET 'Oracle_Key_Vault_password' FOR
CLIENT 'OKV_PASSWORD'
FORCE KEYSTORE IDENTIFIED BY EXTERNAL STORE WITH BACKUP;
```

15. Copy the updated TDE wallet from the primary database host to the standby database hosts, excluding the local auto-login TDE wallet and deleting the obsolete local auto-login TDE wallet on the standby side.

    For example:

```
$ rsync -rvpt --exclude '*.sso' --delete-excluded /etc/ORACLE/KEYSTORES/
finance/tde/ Phoenix:/etc/ORACLE/KEYSTORES/finance/tde/
```

16. On the CDB root of the standby, using SQL*Plus, recreate the local auto-login TDE wallet, and then close it.

```
ADMINISTER KEY MANAGEMENT CREATE LOCAL AUTO_LOGIN KEYSTORE
FROM KEYSTORE IDENTIFIED BY TDE_wallet_password;
ADMINISTER KEY MANAGEMENT SET KEYSTORE CLOSE CONTAINER = ALL;
```

    At this stage, the primary and standby databases are ready to be migrated into Oracle Key Vault.

17. In the primary and standby databases, change the `TDE_CONFIGURATION` dynamic parameter to `OKV|FILE`.

```
ALTER SYSTEM SET TDE_CONFIGURATION = "KEYSTORE_CONFIGURATION=OKV|FILE"
SCOPE = BOTH;
```

18. Run the `ADMINISTER KEY MANAGEMENT…MIGRATE` command.

    This command decrypts the data encryption keys with the most recent key from the TDE wallet, and re-encrypts them with a new TDE master encryption key that is created in Oracle Key Vault. If this is a container database, then each open PDB will go through these

steps. This entails no downtime; the operation lists approximately 10 to 15 seconds for each PDB.

```
ADMINISTER KEY MANAGEMENT SET KEY
IDENTIFIED BY "Oracle_Key_Vault_password"
FORCE KEYSTORE MIGRATE USING "TDE_wallet_password";
```

19. Add the Oracle Key Vault password, and remove the old TDE wallet password, from the local auto-login wallet that enables to replace the TDE wallet password on the SQL*plus command line with `EXTERNAL STORE`.

```
ADMINISTER KEY MANAGEMENT ADD SECRET 'Oracle_Key_Vault_password'
FOR CLIENT "OKV_PASSWORD"
TO LOCAL AUTO_LOGIN KEYSTORE '/etc/ORACLE/KEYSTORES/finance/tde_seps';

ADMINISTER KEY MANAGEMENT DELETE SECRET FOR CLIENT 'TDE_WALLET'
FROM LOCAL AUTO_LOGIN KEYSTORE '/etc/ORACLE/KEYSTORES/finance/tde_seps';
```

20. Now that primary and standby databases have been migrated to Oracle Key Vault, delete the old wallets (only because you have uploaded the keys into Oracle Key Vault in step 13.

```
$ rm -v /etc/ORACLE/KEYSTORES/finance/tde/*
```

21. On the primary and standby databases, create a local auto-login TDE wallet in *WALLET_ROOT*`/tde` that only contains the Oracle Key Vault password to enable an auto-login Oracle Key Vault configuration.

    For example:

```
ADMINISTER KEY MANAGEMENT ADD SECRET 'Oracle_Key_Vault_password'
FOR CLIENT 'OKV_PASSWORD' TO LOCAL AUTO_LOGIN KEYSTORE '/etc/ORACLE/
KEYSTORES/finance/tde';
```

22. Create a tag for each key that is associated with each PDB.

    During migration, Oracle Key Vault created a new key for each PDB, but without a tag. The tag that you create makes it easier to find which key belongs to which PDB. You should give each key a tag that matches the PDB name, and a time stamp.

```
SELECT ' ADMINISTER KEY MANAGEMENT SET TAG '''||SYS_CONTEXT('USERENV',
'CON_NAME')||' '||TO_CHAR (SYS_EXTRACT_UTC (ACTIVATION_TIME),
'YYYY-MM-DD HH24:MI:SS"Z"')||''' FOR '''||KEY_ID||'''
FORCE KEYSTORE IDENTIFIED BY EXTERNAL STORE;' AS "SET TAG COMMAND"
FROM V$ENCRYPTION_KEYS
WHERE CREATOR_PDBNAME = SYS_CONTEXT('USERENV', 'CON_NAME')
ORDER BY CREATION_TIME DESC FETCH FIRST 1 ROWS ONLY;
```

11.2.6 Isolating an Encrypted PDB in an Oracle Data Guard Environment

To isolated PDB's in an Oracle Data Guard environment, you must manually perform the configuration.

This is because the `ADMINISTER KEY MANAGEMENT` command that is run on the primary does not affect the standby. First, you isolate the PDB in the primary database. The `ADMINISTER KEY MANAGEMENT ISOLATE KEYSTORE` command performs the necessary isolation tasks, such as

changing the PDB's TDE_CONFIGURATION parameter to FILE and moving the key from the united mode wallet to the newly created isolated mode wallet. Next, you must perform these same tasks manually on the standby to complete the isolation process.

1. If you have not done so you, log in to the CDB root of the primary database, and then isolate the PDB.

```
ADMINISTER KEY MANAGEMENT ISOLATE KEYSTORE
IDENTIFIED BY "new_isolated_pdb_keystore_password"
FROM ROOT KEYSTORE
IDENTIFIED BY "root_keystore_password"
EXTERNAL STORE WITH BACKUP;
```

2. Log in to the server where the standby database is configured.
3. Ensure that there is a directory for the PDB (for example, $WALLET_ROOT/*pdb_guid*/tde/).
4. Connect to the PDB as a user who has the ALTER SYSTEM privilege.
5. Set the KEYSTORE_CONFIGURATION parameter for this PDB.

```
ALTER SYSTEM SET TDE_CONFIGURATION="KEYSTORE_CONFIGURATION=FILE"
SCOPE=BOTH;
```

6. At the command line, from the CDB root, copy the wallet (.p12 and .sso) from the primary database to the standby; place this wallet in the $WALLET_ROOT/*pdb_guid*/tde/ directory.
7. Close and then re-open the CDB root's wallet.

```
ADMINISTER KEY MANAGEMENT SET KEYSTORE CLOSE
FORCE KEYSTORE IDENTIFIED BY root_wallet_password;

ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN
FORCE KEYSTORE IDENTIFIED BY root_wallet_password;
```

8. Connect to the PDB.
9. Open the wallet in the PDB.

```
ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN
FORCE KEYSTORE IDENTIFIED BY pdb_wallet_password;
```

11.2.7 Uncoupling the Standby Database from the Primary Database Online Encryption Process

You can use the DB_RECOVERY_AUTO_REKEY initialization parameter to control how Transparent Data Encryption (TDE) rekey operations are performed in an Oracle Data Guard environment.

- Set DB_RECOVERY_AUTO_REKEY as follows:
  - ON, which is the default for the standby, enables the standby to automatically perform an online tablespace rekey operation as part of the standby media recovery process. Be aware that this setting pauses media recovery. If the tablespace is large, then you could observe extended standby apply lag. To enable DB_RECOVERY_AUTO_REKEY:

```
ALTER SYSTEM _RECOVERY_AUTO_REKEY = ON;
```

You can can set `DB_RECOVERY_AUTO_REKEY` to `ON` on primary so that media recovery (for example, a restore from backup) will perform this rekey operation. However, this setting can also slow down media recovery and hence, delay primary open operation.

– `OFF`, which is the default for the primary, does not perform a tablespace rekey operation. This setting enables an extended standby database to avoid lag time during an online conversion, and is designed for large Oracle Data Guard tablespace deployments. This enables the standby recovery to only record the tablespace key information but not perform the rekey operation inline. In the `V$ENCRYPTED_TABLESPACES` dynamic view, after the `STATUS` column value for the corresponding tablespace becomes `REKEYING` or `ENCRYPTING` on the standby database, then you can use a standby SQL session to issue an `ALTER TABLESPACE ENCRYPTION ONLINE FINISH REKEY|ENCRYPT` command to perform the rekey in parallel of media recovery. For example, assuming that the `STATUS` column will change to `REKEYING`:

```
ALTER SYSTEM _RECOVERY_AUTO_REKEY = OFF;
...
ALTER TABLESPACE ENCRYPTION ONLINE FINISH REKEY;
```

Related Topics

- *Oracle Database Reference*

11.3 How Transparent Data Encryption Works with Oracle Real Application Clusters

Oracle Real Application Clusters (Oracle RAC) nodes can share both a TDE wallets and an external keystore.

11.3.1 About Using Transparent Data Encryption with Oracle Real Application Clusters

Oracle requires a shared TDE wallet for Oracle Real Application Clusters (Oracle RAC), or a shared common virtual wallet in Oracle Key Vault among cluster instances.

A TDE configuration with Oracle Key Vault uses a network connection from each instance of the database to the external key manager. In Oracle Key Vault, you must create one endpoint for each instance of the Oracle RAC-enabled database, and one virtual wallet for each Oracle RAC-enabled database. Then, make that virtual wallet the default wallet of all endpoints that belong to that database. In an Oracle RAC-enabled Data Guard configuration, all instances (primary and all standby databases) share that one virtual wallet. With this configuration, set key and re-key operations are completely transparent because all participating instances are automatically synchronized. This eliminates the need to manually copy the TDE wallet to each of the other nodes in the cluster.

Oracle does not support the use of individual TDE wallets for each Oracle RAC node. Instead, use shared wallets for TDE in the Oracle RAC environment. This enables all of the instances to access the same shared TDE wallet. If your site uses Oracle Automatic Storage Management Cluster File System (Oracle ACFS), then this is the preferred location for a shared wallet. Directly sharing the wallet in Oracle Automatic Storage Management (Oracle ASM) (the recommended default is `+DATA/$ORACLE_UNQNAME/`) is an alternative if Oracle ACFS is not available..

Keystore operations (such as opening or closing the keystore, or rekeying the TDE master encryption key) can be issued on any one Oracle RAC instance. Internally, the Oracle database takes care of synchronizing the keystore context on each Oracle RAC node, so that the effect of the keystore operation is visible to all of the other Oracle RAC instances in the cluster. Similarly, when a TDE master encryption key rekey operation takes place, the new key becomes available to each of the Oracle RAC instances. You can perform other keystore operations, such as exporting TDE master encryption keys, rotating the keystore password, merging keystores, or backing up keystores, from a single instance only.

When using a shared file system, ensure that the `WALLET_ROOT` static system parameter for all of the Oracle RAC instances point to the same shared TDE wallet location, as follows:

```
ALTER SYSTEM SET WALLET_ROOT = '+DATA/<db-unique-name>' SCOPE = SPFILE SID = '*';
ALTER SYSTEM SET TABLESPACE_ENCRYPTION = 'AUTO_ENABLE' SCOPE = SPFILE SID = '*';

$ srvctl stop database -db DB_NAME -o immediate
$ srvctl start database -db DB_NAME

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

> **Note:**
>
> Storing TDE master encryption keys in individual wallets per Oracle Real Application Clusters (Oracle RAC) instance is not supported. As an alternative, use Oracle Key Vault for centralized key management across your on-premises or Cloud-based database deployments, or Oracle Automatic Storage Management (Oracle ASM), or Oracle ASM Cluster File System (Oracle ACFS) to provide local shared wallets.

11.3.2 Configuring TDE in Oracle Real Application Clusters for Oracle Key Vault

You can configure TDE in Oracle Real Application Clusters (Oracle RAC) on Oracle Exadata Cloud at Customer (ExaCC) and other servers for centralized key management provided by Oracle Key Vault.

The following scenario assumes that you have a multitenant two-node Oracle RAC configuration. In this procedure, you must complete the following steps in the order shown. After you have completed this procedure, the Oracle RAC environment will exclusively use Oracle Key Vault for key management for Transparent Data Encryption.

Before you begin, monitor the alert logs of your running Oracle RAC database. The Java version that is included in the default Oracle Database release 23ai installation can be used to install the Oracle Key Vault client with the RESTful services.

1. Download the Oracle Key Vault deployment script that the Oracle Key Vault administrators prepared to enable database administrators to automatically register their Oracle databases with Oracle Key Vault.

   Oracle Key Vault RESTful Services Administrator's Guide has an example of how to create a script to automatically enroll Oracle databases as endpoints. The deployment scripts reside on a shared file system from with database administrators can download. There are two different versions of these deployment scripts. One script is for only the first node (which is called lead node in this procedure) and the other script is for all other nodes (which are called secondary nodes in this procedure). You can use these scripts for an Oracle RAC or an Oracle Data Guard environment.

Another component that the Oracle Key Vault administrators prepare and add to the deployment script is a configuration file that contains all details for the deployment scripts to connect to Oracle Key Vault.

2. Copy the two deployment scripts (`primary.zip` and `secondary.zip`) that an Oracle Key Vault administrator created for database administrators to download from a shared location.

   a. Copy the `primary.zip` file to the lead node.

   ```
   $ scp user@ip_address:/path/to/file/primary.zip .
   ```

   b. Copy the `secondary.zip` file to each secondary node.

   ```
   $ scp user@ip_address:/path/to/file/secondary.zip .
   ```

3. Extract the zip files.

   a. **On the lead node:** Extract the `primary.zip` file.

   ```
   $ unzip primary.zip
   ```

   b. **On the secondary nodes:** Extract the `secondary.zip` file.

   ```
   $ unzip secondary.zip
   ```

4. Create the following directories on all nodes:

   For example:

   ```
   $ mkdir -pv /u01/opt/oracle/product/okv
   $ mkdir -pv /u01/opt/oracle/product/tde
   $ mkdir -pv /u01/opt/oracle/product/tde_seps
   ```

   In this specification:

   - The `/u01/opt/oracle/product` directory will be defined as `WALLET_ROOT` in a later step.
   - `/u01/opt/oracle/product/okv` is the installation directory for the Oracle Key Vault client software. Depending on how the `TDE_CONFIGURATION` parameter is set, the Oracle Database will look for the Oracle Key Vault client software in *`wallet_root/okv`*.
   - `/u01/opt/oracle/product/tde` will store an auto-login wallet, which only contains the future Oracle Key Vault password, enabling an auto-login Oracle Key Vault configuration. Depending on how `TDE_CONFIGURATION` is set, the Oracle Database will look for the TDE wallet or an auto-open wallet for Oracle Key Vault, in *`wallet_root/tde`*.
   - `/u01/opt/oracle/product/tde_seps` will store an auto-login wallet, which only contains the future Oracle Key Vault password. This will hide the Oracle Key Vault password from the SQL*Plus command line and potentially from the database administrator to enforce separation of duties between Oracle database administrators and Oracle Key Vault administrators.

5. Run the `primary-run-me.sh` and `secondary-run-me.sh` scripts, which contain the commands for the RESTful API to run in Oracle Key Vault.

   The Oracle Key Vault RESTful services will run these commands in order to register this database in Oracle Key Vault with unique wallet and endpoint names.

   a. **On the lead node:** This script creates a shared wallet (for the lead and all secondary nodes) and an endpoint in Oracle Key Vault, associates this endpoint for the lead node with the shared wallet, and downloads and installs the Oracle Key Vault client into an

existing installation directory. With the `WALLET_ROOT` configuration, this directory is *wallet_root*`/okv`.

```
$ more primary-run-me.sh

#!/bin/bash
export EP_NAME=${ORACLE_SID^^}_on_$(hostname -s)
export WALLET_NAME=${ORACLE_UNQNAME^^}${HOSTNAME//[!0-9]/}
curl -Ok https://192.168.1.181:5695/okvrestclipackage.zip --tlsv1.2
unzip -Voj okvrestclipackage.zip lib/okvrestcli.jar -d ./lib
cat > /home/oracle/deploy-OKV.sh << EOF
#!/bin/bash
okv manage-access wallet create --wallet ${WALLET_NAME} --unique FALSE
okv admin endpoint create --endpoint ${EP_NAME} --description "$(hostname -f) $
(hostname -i)" --subgroup "USE CREATOR SUBGROUP" --unique FALSE
okv admin endpoint update --endpoint ${EP_NAME} --strict-ip-check TRUE
okv manage-access wallet set-default --wallet ${WALLET_NAME} --endpoint $
{EP_NAME}
expect << _EOF
    set timeout 120
    spawn okv admin endpoint provision --endpoint ${EP_NAME} --location /etc/
ORACLE/KEYSTORES/${ORACLE_UNQNAME^^}/okv --auto-login FALSE
    expect "Enter Oracle Key Vault endpoint password: "
    send "change-on-install\r"
    expect eof
_EOF
EOF
```

b. **Secondary nodes:** This script only creates an endpoint for the secondary nodes, associates the endpoint of the secondary nodes with the shared wallet, and downloads and installs the Oracle Key Vault client into the **existing** installation directory on each secondary node.

```
$ more run-me.sh

#!/bin/bash
export EP_NAME=${ORACLE_SID^^}_on_$(hostname -s)
export WALLET_NAME=${ORACLE_UNQNAME^^}${HOSTNAME//[!0-9]/}
curl -Ok --tlsv1.2 https://<OKV-IP-addr>:5695/okvrestclipackage.zip
unzip -Voj okvrestclipackage.zip lib/okvrestcli.jar -d ./lib
cat > /home/oracle/deploy-OKV.sh << EOF
#!/bin/bash
okv admin endpoint create --endpoint ${EP_NAME} --description "$(hostname -f) $
(hostname -i)" --subgroup "USE CREATOR SUBGROUP" --unique FALSE
okv admin endpoint update --endpoint ${EP_NAME} --strict-ip-check TRUE
okv manage-access wallet set-default --wallet ${WALLET_NAME} --endpoint $
{EP_NAME}
expect << _EOF
    set timeout 120
    spawn okv admin endpoint provision --endpoint ${EP_NAME} --location /etc/
ORACLE/KEYSTORES/${ORACLE_UNQNAME^^}/okv --auto-login FALSE
    expect "Enter Oracle Key Vault endpoint password: "
    send "change-on-install\r"
    expect eof
_EOF
EOF
```

6. After successful installation of the Oracle Key Vault client, run the `root.sh` script to install the PKCS library on all nodes.

```
# Oracle_Key_Vault_installation_directory/bin/root.sh
```

The following output should appear:

```
Creating directory: /opt/oracle/extapi/64/hsm/oracle/1.0.0/
Copying PKCS library to /opt/oracle/extapi/64/hsm/oracle/1.0.0/
Setting PKCS library file permissions
```

7. Run the Oracle Key Vault `okvutil changepwd` command on all nodes to change the password for the Oracle Key Vault client that you installed.

   Because all database administrators downloaded the same deployment script, all databases have the same password into Oracle Key Vault. This step enables each database to have a unique password.

   ```
   $ /u01/opt/oracle/product/okv/bin/okvutil changepwd -t wallet -l /u01/opt/oracle/
   product/okv/ssl/

   Enter wallet password: default_password
   Enter new wallet password: Oracle_Key_Vault_password
   Confirm new wallet password: Oracle_Key_Vault_password
   Wallet password changed successfully
   ```

8. On all nodes, add the Oracle Key Vault password into a local auto-login wallet to hide the newly changed password from database administrators.

   ```
   sqlplus c##sec_admin as syskm
   Enter password: password

   ADMINISTER KEY MANAGEMENT ADD SECRET 'Oracle_Key_Vault_password'
   FOR CLIENT 'OKV_PASSWORD'
   TO LOCAL AUTO_LOGIN KEYSTORE '/u01/opt/oracle/product/tde_seps';
   ```

9. In the root container, run the `ALTER SYSTEM` statement to set the static `WALLET_ROOT` parameter to configure the encryption wallet location for all instances:

   ```
   CONNECT / AS SYSDBA

   ALTER SYSTEM SET TABLESPACE_ENCRYPTION = 'AUTO_ENABLE' SCOPE = SPFILE SID = '*';
   ALTER SYSTEM SET WALLET_ROOT = '/u01/opt/oracle/product/' SCOPE = SPFILE SID = '*';
   ```

10. Restart the database.

11. In the root container, use the `ALTER SYSTEM` statement to set the dynamic `TDE_CONFIGURATION` parameter.

    For example:

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

12. In Oracle Database 23ai, `AES256` with XTS is the default; it is not recommended to change the cipher mode or the encryption key length.

13. In the root container, open the keystore, which opens the connection to Oracle Key Vault for the root container and all open PDBs.

    Note that the Oracle Key Vault password has been replaced in all subsequent `ADMINISTER KEY MANAGEMENT` commands with `EXTERNAL STORE`, because the database automatically retrieves the Oracle Key Vault password from the local auto-login wallet in `<WALLET_ROOT>/tde_seps` that you created earlier.

    ```
    ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN
    IDENTIFIED BY EXTERNAL STORE
    CONTAINER = ALL;
    ```

14. In the root container, set the master encryption key.

```
ADMINISTER KEY MANAGEMENT SET KEY
IDENTIFIED BY EXTERNAL STORE
CONTAINER = CURRENT;
```

15. Create and activate a tagged master encryption key in all PDBs in this container.

    The benefit of adding tagged master encryption keys to PDBs is that it enables you to easily identify keys that belong to a certain PDB.

    a. Connect to each PDB and run the following `SELECT` statement to create an `ADMINISTER KEY MANAGEMENT` command that contains the PDB name and time stamp as a tag for the PDB's master encryption key.

    ```
    SELECT ' ADMINISTER KEY MANAGEMENT SET KEY
    USING TAG '''||SYS_CONTEXT('USERENV', 'CON_NAME')||' '||
    TO_CHAR (SYSDATE, 'YYYY-MM-DD HH24:MI:SS')||'''
    IDENTIFIED BY EXTERNAL STORE;' "SET KEY COMMAND" FROM DUAL;
    ```

    b. Run the generated output of this `SELECT` statement.

    For example:

    ```
    ADMINISTER KEY MANAGEMENT SET KEY
    USING TAG 'pdb_name date time'
    IDENTIFIED BY EXTERNAL STORE;
    ```

16. On all nodes, add the Oracle Key Vault password into an auto-login wallet to enable auto-login connection into Oracle Key Vault.

    This step is mandatory in Oracle RAC. Having an auto-login connection into Oracle Key Vault is especially important when Oracle RAC nodes are automatically restarted (for example, while applying quarterly release upgrades using the `opatchauto` patch tool).

    ```
    ADMINISTER KEY MANAGEMENT ADD SECRET 'Oracle_Key_Vault_password'
    FOR CLIENT 'OKV_PASSWORD'
    TO LOCAL AUTO_LOGIN KEYSTORE '/u01/opt/oracle/product/tde';
    ```

17. In the root container, run the `ALTER SYSTEM` statement to change the `TDE_CONFIGURATION` parameter.

    For example:

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

18. From the root, encrypt sensitive credential data with `AES256` for database links in the `SYS.LINK$` and `SYS.SCHEDULER$_CREDENTIAL` system tables.

    This command requires the `SYSKM` administrative privilege:

    ```
    sqlplus c##sec_admin as syskm
    Enter password: password

    ALTER DATABASE DICTIONARY ENCRYPT CREDENTIALS;
    ```

19. Log in to the PDB and create a tablespace.

    For example, to create a tablespace named `protected`:

    ```
    CREATE TABLESPACE protected DATAFILE SIZE 50M;
    ```

20. Confirm that the tablespace is encrypted even though the encryption clauses were omitted.

    ```
    SELECT C.NAME AS pdb_name, T.NAME AS tablespace_name, E.ENCRYPTIONALG AS ALG
    FROM V$TABLESPACE T, V$ENCRYPTED_TABLESPACES E, V$CONTAINERS C
    WHERE E.TS# = T.TS# AND E.CON_ID = T.CON_ID AND E.CON_ID = C.CON_ID
    ORDER BY E.CON_ID, T.NAME;
    ```

21. Create a table in the encrypted tablespace that you just created.

    For example:

    ```
    CREATE TABLE SYSTEM.test TABLESPACE protected
    AS SELECT * FROM DBA_OBJECTS;
    ```

22. Select from this table to confirm that you can read encrypted data:

    ```
    SELECT COUNT(*), OWNER FROM SYSTEM.test
    GROUP BY OWNER
    ORDER BY 1 DESC;
    ```

23. In the PDBs, encrypt the existing tablespaces.

    Optionally, encrypt the `SYSTEM`, `SYSAUX`, and `USERS` tablespaces. If you omit the encryption algorithm, then the default algorithm (`AES128`, or the algorithm that you specified earlier) is applied.

    ```
    ALTER TABLESPACE tablespace_name ENCRYPTION ONLINE ENCRYPT;
    ```

24. Optionally, validate the configuration.

    a. Confirm that the auto-login for Oracle Key Vault is working.

       You can test this by restarting the database, logging into the PDB, and then selecting from the encrypted table. To restart the database:

       ```
       $ srvctl stop database -db database_name -o immediate
       $ srvctl start database -db database_name
       ```

       After logging in to the PDB, select from the `SYSTEM.test` table.

       ```
       SELECT COUNT(*), OWNER FROM SYSTEM.test
       GROUP BY OWNER
       ORDER BY 1 DESC;
       ```

    b. Confirm that the master encryption key re-key operations in all open PDBs are successful.

       First, as a user who has the `SYSKM` administrative privilege, run the following `SELECT` statement to create an `ADMINISTER KEY MANAGEMENT` command that contains the PDB name and time stamp.

       ```
       SELECT ' ADMINISTER KEY MANAGEMENT SET KEY
       USING TAG '''||SYS_CONTEXT('USERENV', 'CON_NAME')||' '||
       TO_CHAR (SYSDATE, 'YYYY-MM-DD HH24:MI:SS')||'''
       FORCE KEYSTORE IDENTIFIED BY EXTERNAL STORE;' "RE-KEY COMMAND";
       ```

       Next, run the generated output of this `SELECT` statement.

       ```
       ADMINISTER KEY MANAGEMENT SET KEY
       USING TAG 'pdb_name date time'
       FORCE KEYSTORE IDENTIFIED BY EXTERNAL STORE;
       ```

    c. From the root container, re-key previously encrypted sensitive credential data in the `SYS.LINK$` and `SYS.SCHEDULER$_CREDENTIAL` system tables.

       This command requires the `SYSKM` administrative privilege:

       ```
       ALTER DATABASE DICTIONARY REKEY CREDENTIALS;
       ```

    d. Drop the `protected` tablespace and its table, `test`.

       ```
       DROP TABLESPACE protected
       INCLUDING CONTENTS AND DATAFILES;
       ```

Related Topics

- Supported Encryption and Integrity Algorithms
  Oracle supports the AES, ARIA and DES algorithms.

11.4 How Transparent Data Encryption Works with SecureFiles

SecureFiles, which stores LOBS, has three features: compression, deduplication, and encryption.

11.4.1 About Transparent Data Encryption and SecureFiles

SecureFiles encryption uses TDE to provide the encryption facility for LOBs.

When you create or alter tables, you can specify the SecureFiles encryption or LOB columns that must use the SecureFiles storage. You can enable the encryption for a LOB column by either using the current Transparent Data Encryption (TDE) syntax or by using the `ENCRYPT` clause as part of the LOB parameters for the LOB column. The `DECRYPT` option in the current syntax or the LOB parameters turn off encryption.

11.4.2 Example: Creating a SecureFiles LOB with a Specific Encryption Algorithm

The `CREATE TABLE` statement can create a SecureFiles LOB with encryption specified.

Example 11-1 shows how to create a SecureFiles LOB in a `CREATE TABLE` statement.

Example 11-1 Creating a SecureFiles LOB with a Specific Encryption Algorithm

```
CREATE TABLE table1 ( a BLOB ENCRYPT USING 'AES256')
    LOB(a) STORE AS SECUREFILE (
    CACHE
    );
```

11.4.3 Example: Creating a SecureFiles LOB with a Column Password Specified

The `CREATE TABLE` statement can create a SecureFiles LOB with a column password.

Example 11-2 shows an example of creating a SecureFiles LOB that uses password protections for the encrypted column.

All of the LOBS in the LOB column are encrypted with the same encryption specification.

Example 11-2 Creating a SecureFiles LOB with a Column Password Specified

```
CREATE TABLE table1 (a VARCHAR2(20), b BLOB)
    LOB(b) STORE AS SECUREFILE (
        CACHE
        ENCRYPT USING 'AES192' IDENTIFIED BY password
    );
```

11.5 How Transparent Data Encryption Works with Oracle Call Interface

Transparent Data Encryption does not have any effect on the operation of Oracle Call Interface (OCI).

For most practical purposes, TDE is transparent to OCI except for the row shipping feature. You cannot use the OCI row shipping feature with TDE because the key to make the row usable is not available at the receipt-point.

11.6 How Transparent Data Encryption Works with Editions

Transparent Data Encryption does not have any effect on the Editions feature of Oracle Database.

For most practical purposes, TDE is transparent to Editions. Tables are always noneditioned objects. TDE Column Encryption encrypts columns of the table. Editions are not affected by TDE tablespace encryption.

11.7 Configuring Transparent Data Encryption to Work in a Multidatabase Environment

Each Oracle database on the same server (such as databases sharing the same Oracle binary but using different data files) must access its own TDE keystore.

Keystores are not designed to be shared among databases. By design, there must be one keystore per database. You cannot use the same keystore for more than one database.

- To configure the use of keystores in a multidatabase environment, use one of the following options:
  - **Option 1:** Specify the keystore location by individually setting the `WALLET_ROOT` static initialization parameter and the `TDE_CONFIGURATION` dynamic initialization parameter (its `KEYSTORE_CONFIGURATION` attribute set to `FILE`) for each CDB (or standalone database).

    For example:

    ```
    WALLET_ROOT = '$ORACLE_BASE/admin/${ORACLE_SID}' scope = spfile;

    TDE_CONFIGURATION="KEYSTORE_CONFIGURATION=FILE"
    ```

    Restart the database for the parameter to take effect.

> **Caution:**
>
> Using a keystore from another database can cause partial or complete data loss.

Related Topics

- Transparent Data Encryption Keystore Search Order
  The search order for the TDE keystore depends on how you have set either the instance initialization parameters, the `sqlnet.ora` parameters, or the environment variables.
- *SQL*Plus User's Guide and Reference*

12

Frequently Asked Questions About Transparent Data Encryption

Users frequently have questions about transparency and performance issues with Transparent Data Encryption.

12.1 Transparency Questions About Transparent Data Encryption

Transparent Data encryption handles transparency in data in a variety of ways.

Security auditors occasionally ask detailed questions about the encryption used by Transparent Data Encryption (TDE). They request information about TDE keys, algorithms, lengths, and keystores and then directly compare to requirements of regulations such as PCI-DSS. This topic contains important details about TDE encryption and key management. This information is current as of Oracle Database 12*c* (12.1.0.2). It is intended to help TDE customers respond to auditor questions quickly and accurately.

1. **Is Transparent Data Encryption compatible with my application software?**

   Transparent Data Encryption is compatible with applications by default because it does not alter the inbound SQL statements or the outbound SQL query results. Oracle runs internal testing and validation of certain Oracle and third-party application software to capture helpful deployment tips or scripts, and to evaluate performance profiles.

   Be aware of the difference between Transparent Data Encryption and the `DBMS_CRYPTO` PL/SQL package. This package is intended for different customer use cases. It is an API and toolkit solution and as such, it is non-transparent.

2. **Is Transparent Data Encryption compatible with other Oracle Database tools and technologies that I am using?**

   One of the chief benefits of Transparent Data Encryption is its integration with frequently used Oracle Database tools and technologies such as high-availability clusters, storage compression, backup compression, data movement, database backup and restore, and database replication. Specific Oracle technologies that are integrated directly with Transparent Data Encryption include Oracle Real Application Clusters (Oracle RAC), Oracle Recovery Manager (RMAN), Oracle Data Guard, Advanced Compression, Oracle Data Pump, and Oracle GoldenGate, among others. Transparent Data Encryption also has special points of integration with Oracle Exadata that fully use unique features of Oracle-engineered systems.

   Transparent Data Encryption also works easily with security features of the Oracle Database. With Transparent Data Encryption, privilege grants, roles, Oracle Database Vault realms, Virtual Private Database policies, and Oracle Label Security labels remain in effect. You can use these and other security features in tandem with Transparent Data Encryption encryption.

3. **Are there any known Transparent Data Encryption limitations or incompatibilities?**

   - **TDE column encryption:** TDE column encryption encrypts and decrypts data transparently when data passes through the SQL layer. Some features of Oracle will

bypass the SQL layer, and hence cannot benefit from TDE column encryption. The following are known database features that TDE column encryption does not support, and their relevant software version numbers:

- Materialized View Logs (not supported prior to Oracle Database 11g release 2)
- Synchronous and asynchronous change data capture for data warehousing (CDC)
- Transportable Tablespaces
- LOBs

Note that Secure Files were introduced in Oracle Database 11g release 1, so it is not supported with TDE column encryption prior to that release

- **TDE tablespace encryption:** TDE tablespace encryption encrypts all content that is stored in the tablespace at the block level in storage, and it generally does not conflict with other database features. TDE tablespace encryption does not have any of the limitations that TDE column encryption has. However, you can use full transportable tablespaces (TTS) with Oracle Data Pump compression and encryption when going from a TDE-encrypted source to a TDE-encrypted destination. You must have an Oracle Database release 12*c* or later database instance available so that you can use its key export or keystore (wallet) merge capabilities to get the correct master encryption key to the destination database host without having to overwrite the original Oracle wallet file. This process is subject to the standard TTS limitations, and you must remember to check for compatible endianness.

4. **What types of keys and algorithms does TDE use?**

   TDE relies on two distinct sets of encryption keys. The first set of encryption keys are TDE tablespace encryption keys, which are used to transparently encrypt and decrypt stored data. DEKs are generated automatically by the database, stored internally in the database in encrypted form, and managed mostly behind the scenes. One place where end-users interact with DEKs is when selecting the encryption algorithm and key length that TDE will use, which can be `3DES168`, `AES128`, `AES192`, or `AES256`. This selection is made independently for each table containing encrypted columns and for each encrypted tablespace. You may also hear DEKs referred to as table keys (column encryption) or tablespace keys (tablespace encryption). The table keys can be used in cipher block chaining (CBC) or Galois/Counter (GCM) mode, and the tablespace keys can be used in used in cipher feedback mode (CFB) or tweakable block ciphertext stealing (XTS) operating mode.

   The second set of encryption keys consists of current and historical key encryption keys (KEK), also known as master encryption keys. The master encryption keys are generated automatically by the database, used automatically to encrypt and decrypt DEKs as needed, and stored externally in a protected keystore. Users may interact with the current master encryption key by periodically rekeying it, modifying certain key attributes, and so forth. Typically, the keystore for master encryption keys is either an Oracle wallet (out-of-the-box solution) or Oracle Key Vault (a specialized key management product). Although the database uses only one TDE master key at a time, all rekeyed master encryption keys are retained in the keystore for long-term recovery of encrypted data backups.Master encryption keys always are AES256. They encrypt and decrypt DEKs using CBC operating mode. For both DEKs and master encryption keys, the underlying key material is not directly exposed. End-users see only attributes of keys necessary to manage TDE.

5. **How are Oracle keystores containing master encryption keys protected?**

   There are three different types of keystore to consider when you use an Oracle wallet as the keystore for master encryption keys: password-based, auto-login, and local auto-login. All of these keystore externalize master encryption keys, so they are separate from

TDE-encrypted data. Oracle recommends that you place wallet files in local or network directories that are protected by tight file permissions and other security measures.

The password-based wallet is an encrypted key storage file (`ewallet.p12`) that follows the PKCS #12 standard. It is encrypted by a password-derived key according to the PKCS #5 standard. A human user must enter a command containing the password for the database to open the wallet, decrypt its contents, and gain access to keys. The password-based wallet is the default keystore for TDE master keys. In the past, it was encrypted using the 3DES168 encryption algorithm and CBC operating mode. The `orapki` command `convert wallet` enables you to convert password-based wallets to AES256 and CBC operating mode. For example, to use the `compat_v12` setting to perform the conversion from 3DES to AES256:

```
orapki wallet convert -wallet wallet_location [-pwd password] [-compat_v12]
```

Auto-login wallets (`cwallet.sso`) optionally are derived from standard password-based wallets for special cases where automatic startup of the database is required with no human interaction to enter a wallet password. When using auto-login wallet, the master password-based wallet must be preserved because it is needed to rekey the master encryption key. In addition to the best practice of storing auto-login wallet in a local or network directory that is protected by tight file permissions, the file contents are scrambled by the database using a proprietary method for added security. A slight variation on the auto-login wallet called local auto-login wallet has similar behavior. One notable difference with local auto-login wallet is that its contents are scrambled using additional factors taken from the host machine where the file was created. This renders the local auto-login wallet unusable on other host machines. Details of the host factors and scrambling technique are proprietary.

6. **What is Oracle Key Vault and how does it manage TDE master keys?**

   Oracle Key Vault centrally manages TDE master keys, Oracle wallets, Java keystores, and more. It helps you to take control of proliferating keys and key storage files. It includes optimizations specifically for TDE and other components of the Oracle stack.

Related Topics

- *Oracle Database Security Guide*

12.2 Performance Questions About Transparent Data Encryption

There are several performance issues to consider when using Transparent Data Encryption.

1. **What is the typical performance overhead from Transparent Data Encryption?**

   There are many different variables involved in the creation of an accurate Transparent Data Encryption performance test. The results can vary depending on the test environment, test case or workload, measurement metrics or methods, and so on. Oracle cannot guarantee a specific performance overhead percentage that can apply in all possible scenarios. In practice, the performance tests by many Transparent Data Encryption customers are often in the low single digits as a percentage, but that is not universally the case.

   If possible, use Oracle Real Application Testing (Oracle RAT) to capture a real production workload and then replay it against Transparent Data Encryption to get a true indication of the performance overhead that the you can expect within your environment.

2. **How can I tune for optimal Transparent Data Encryption performance?**

   - **TDE column encryption:**

  - Limit the crypto processing by only encrypting the subset of columns that are strictly required to be protected. In addition, turn off the optional integrity checking feature.
  - After you apply column encryption, rebuild the column indexes.

- **TDE tablespace encryption:** TDE tablespace encryption improves performance by caching unencrypted data in memory in the SGA buffer cache. This feature reduces the number of crypto operations that must be performed when users run `SELECT` queries, which draw from the SGA instead of drawing from disk. (Drawing from disk forces the database to perform decrypt operations.) Ensure that the size of the SGA buffer cache is large enough to take full advantage of this performance optimization.

  Another major performance boost comes from using hardware and software that supports CPU-based cryptographic acceleration available in Intel AES-NI and Oracle SPARC T4/T5. To take advantage of this feature, you must be running a recent version of the database, have a recent version of the operating system installed, and be using hardware that includes crypto acceleration circuitry within its CPUs/cores.

  Database compression further speeds up Transparent Data Encryption performance because the crypto processing occurs on data that already is compressed, resulting in less total data to encrypt and decrypt.

- **In general:**
  - Ensure that you have applied the latest patches, which you can download from My Oracle Support at

    `https://support.oracle.com/portal/`
  - When you specify an encryption algorithm, remember that AES is slightly faster than 3DES. Use AES128 where possible. Be aware that the performance benefit is small.
  - Use Exadata (described in *Oracle Database Testing Guide*), which includes additional performance benefits.

3. **Are there specific issues that may slow down TDE performance, and if so, how do I avoid them?**

   TDE tablespace performance is slower if the database cannot use CPU-based hardware acceleration on the host machine due to factors such as older hardware, an older database version, or an older operating system.

   Note the following with regard to specific database workloads:

   - **Encrypting the whole data set at once (for example, while doing “Bulk Data Load" into an Oracle data warehouse):** Lower crypto performance has been observed during bulk load of new data into the database or data warehouse. New data cannot be cached in SGA, so TDE tablespace encryption performance optimizations are bypassed. Hence, Transparent Data Encryption has no bonus performance benefits in this type of operation.

     Follow these guidelines:

     - Ensure that the database is running on servers with CPU-based cryptographic acceleration. This accelerates not only decrypt operations, but also encrypt operations as well (for loading new data). Take the crypto processing out of band by pre-encrypting the data set and then using Transportable Tablespaces (TTS) to load into the database. Try to parallelize this procedure where possible. This requires the database instance to copy the required TDE key to the keystore on the destination database. The procedure may not be feasible when there is a fixed time window for encryption and loading, and these must be done serially.

  - Consider using TDE column encryption. Encrypt only the handful of sensitive regulated columns instead of encrypting an entire tablespace.

- **Decrypting an entire data set at once (for example, while performing a full table scan by reading directly from disk, with no reading from SGA):**

  Lower crypto performance is observed when running full table scan queries where data is read directly from storage. Certain performance optimizations of TDE tablespace encryption are bypassed (no caching). Hence, Transparent Data Encryption has no bonus performance benefits in this type of operation.

  Follow these guidelines:

  - Ensure that the database is running on servers with CPU-based cryptographic acceleration.
  - Retest the full table scan queries with a larger SGA size to measure performance when data is read from cache. Try setting the Oracle event number 10949 to disable direct path read.
  - Partition the database so that less data is scanned by full table scan operations. Production databases often use partitioning for this kind of scenario (that is, to limit the total amount of data scanned).
  - Consider using TDE column encryption. Encrypt only the handful of sensitive regulated columns instead of encrypting an entire tablespace.

Related Topics

- Performance Overhead of Transparent Data Encryption
  Transparent Data Encryption tablespace encryption has small associated performance overhead.
- *Oracle Database Testing Guide*
- *Oracle Database Testing Guide*

Glossary

Index

Symbols

A

C

D

E

H

I

K

M

O

P

R

S

T

U

V

W