Field Level Encryption Guide in Oracle DB

This guide is to help developers quickly understand the steps needed to implement field level encryption in Oracle DB. Field level encryption is enabled via the *TDE Column Encryption* feature of Oracle DB.

TDE Column Encryption will encrypt sensitive columns such as medical or credit card information.

Identifying the Crown Jewel

Customer ID	**Customer Name**	**Customer Credit Card Number (Crown Jewel)**
123	Jim Halpert	4111111111111111
456	Kevin Malone	3566002020360505
789	Pamela Beasly	371449635398431

The first step is to identify the sensitive data (referred to as Crown Jewels).
Some examples are credit card info, medical records etc.

Customer ID	**Customer Name**	**Customer Credit Card Number (Crown Jewel)**
123	Jim Halpert	MTIzNC01NjctODkwMQ==
456	Kevin Malone	OTk5OS04ODg4LTc3Nz1==
789	Pamela Beasly	OTg3LTY1NC0zMj4QpR1E=

After enabling TDE Column Encryption, only the sensitive column will be encrypted.
To access the plaintext value, access to the appropriate keys is needed.

Restrictions on TDE Column Encryption

In Oracle, TDE Column Encryption cannot be used with the following:

- Columns used in foreign key constraints
- Identity columns
- Indexes (other than B-tree)
- Range scan search via indexing

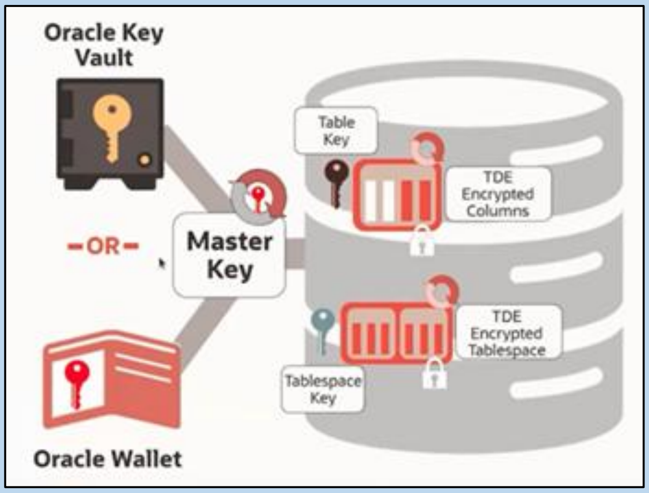
Please refer here for more details.

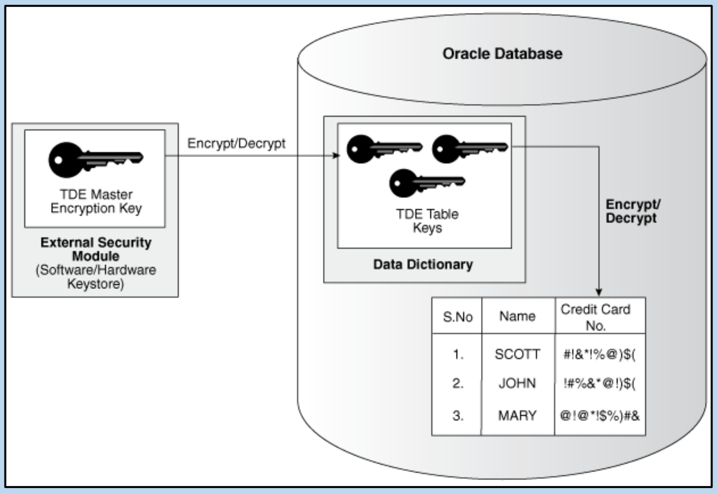
Managing the Keys

There are 2 keys needed:

1. TDE Master Key - Encrypts the Column Encryption Keys
2. TDE Table Key - Encrypts the actual data columns and is encrypted by the CMK.

Note: In Oracle, these 2 keys are stored in a key store, such as HSM or Key Vault. For this guide, Oracle Wallet is used as an example.




Oracle Database
TDE Master Encryption Key
External Security Module (Software/Hardware Keystore)
Encrypt/Decrypt
TDE Table Keys
Data Dictionary
Encrypt/ Decrypt

S.No	Name	Credit Card No.
1.	SCOTT	#!&*!%@)$(
2.	JOHN	!#%&*@!)$(
3.	MARY	@!@*!$%)#&

Requesting Keys

1. Identify column that requires encryption.
2. Raise a ServiceNow request to DBA for enabling encryption on this column.
3. DBA should generate the keys and inform of completion.
4. Run test queries and verify decryption of the sensitive data
    a. Note: Since the encryption is transparent, no configuration is needed from the application level. The database ensures access to the encryption keys via the Oracle Wallet/Keystore, which must be accessible and open during database operations..

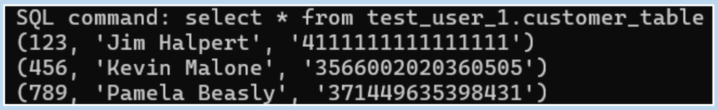
Querying Encrypted Field

To query the encrypted data, we have to ensure the database has access to the keys. There is no configuration needed on the application-level, as the encryption is transparent.

Below is a simple Python example, which is used to access a *customer_table*, where the column *'credit_card_number'* is encrypted.

```
import oracledb

un = 'test_user_1'
cs = 'localhost:1521/XE'
pw = 'password'
with oracledb.connect(user=un, password=pw, dsn=cs) as connection:
    with connection.cursor() as cursor:
        sql = """select * from test_user_1.customer_table"""
        print(f"SQL command: {sql}")

        for r in cursor.execute(sql):
            print(r)
```

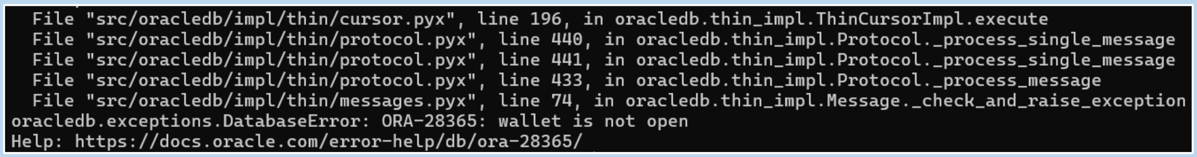
Output when wallet is open



```
SQL command: select * from test_user_1.customer_table
(123, 'Jim Halpert', '4111111111111111')
(456, 'Kevin Malone', '3566002020360505')
(789, 'Pamela Beasly', '371449635398431')
```

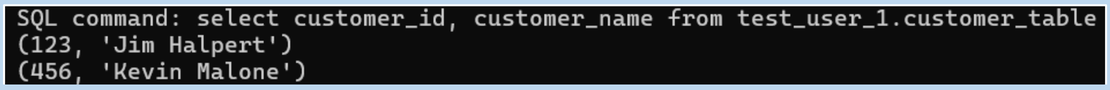
When the wallet is open, the application will retrieve plaintext values for encrypted columns. If the wallet is closed, the application will throw an error for encrypted columns, but non-encrypted columns remain accessible.

Output when wallet is closed



```
  File "src/oracledb/impl/thin/cursor.pyx", line 196, in oracledb.thin_impl.ThinCursorImpl.execute
  File "src/oracledb/impl/thin/protocol.pyx", line 440, in oracledb.thin_impl.Protocol._process_single_message
  File "src/oracledb/impl/thin/protocol.pyx", line 441, in oracledb.thin_impl.Protocol._process_single_message
  File "src/oracledb/impl/thin/protocol.pyx", line 433, in oracledb.thin_impl.Protocol._process_message
  File "src/oracledb/impl/thin/messages.pyx", line 74, in oracledb.thin_impl.Message._check_and_raise_exception
oracledb.exceptions.DatabaseError: ORA-28365: wallet is not open
Help: https://docs.oracle.com/error-help/db/ora-28365/
```

However, the fields which are not encrypted can still be accessed.

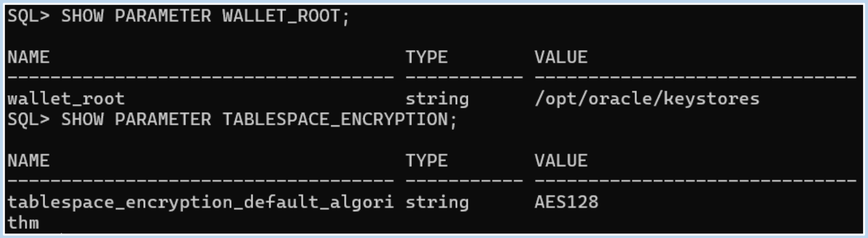


```
SQL command: select customer_id, customer_name from test_user_1.customer_table
(123, 'Jim Halpert')
(456, 'Kevin Malone')
```

Enabling Encryption – For DBA

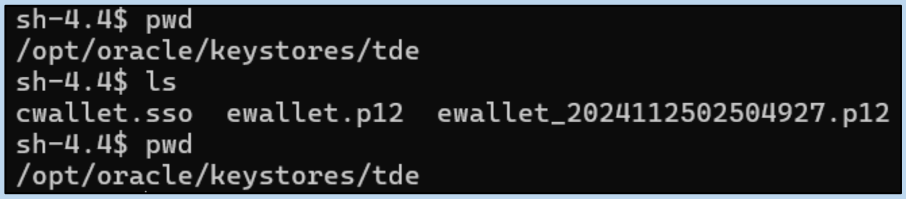
Configuring Keystore Location

1. **It is strongly recommended to use an external volume or a secure, external key management system (e.g., Oracle Key Vault, HSM)** to safeguard encryption keys even if the database server is compromised. For a comparison of the various keystores available, please see here.
2. For the purpose of this guide, a local directory on the database server is used for demonstration purposes only.
3. Using local directory for Oracle Wallet (`/opt/oracle/keystores`) and setting necessary permissions:
    a. `mkdir /opt/oracle/keystores`
    b. `chown -Rv oracle:oinstall /opt/oracle/keystores`
    c. `chmod -Rv 700 /opt/oracle/keystores`
4. Login to the DB with DBA permissions and set the WALLET_ROOT location:
    a. `sqlplus / as sysdba`
    b. `alter system set WALLET_ROOT = '/opt/oracle/keystores' SCOPE=SPFILE;`
    c. `shutdown immediate;`
    d. `startup;`
    e. `SHOW PARAMETER WALLET_ROOT;`
    f. `SHOW PARAMETER TABLESPACE_ENCRYPTION;`
    g. The following parameter values should be visible:



```
SQL> SHOW PARAMETER WALLET_ROOT;

NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
wallet_root                          string      /opt/oracle/keystores
SQL> SHOW PARAMETER TABLESPACE_ENCRYPTION;

NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
tablespace_encryption_default_algori string      AES128
thm
```

    h. Note that the default encryption algorithm used is AES128. This can be changed when specifying the encrypted columns.
5. Next, we need to create the TDE wallet, which will be used to store the keys. Access to this wallet will be controlled by a wallet password.
    a. `alter system set TDE_CONFIGURATION = "KEYSTORE_CONFIGURATION=FILE" scope = both;`
    b. `shutdown immediate;`
    c. `startup;`
    d. `administer key management CREATE KEYSTORE identified by <INSERT-WALLET-PASSWORD>;`
    e. `administer key management CREATE LOCAL AUTO_LOGIN KEYSTORE from keystore identified by <INSERT-WALLET-PASSWORD>;`
    f. `administer key management SET KEY force keystore identified by <INSERT-WALLET-PASSWORD> with backup container = current;`
    g. Note: In this example, the DB is configured to automatically login to the keystore during startup. This can be configured to be manual instead.
    h. The following '*tde*' directory and wallet files would be created in the directory location:



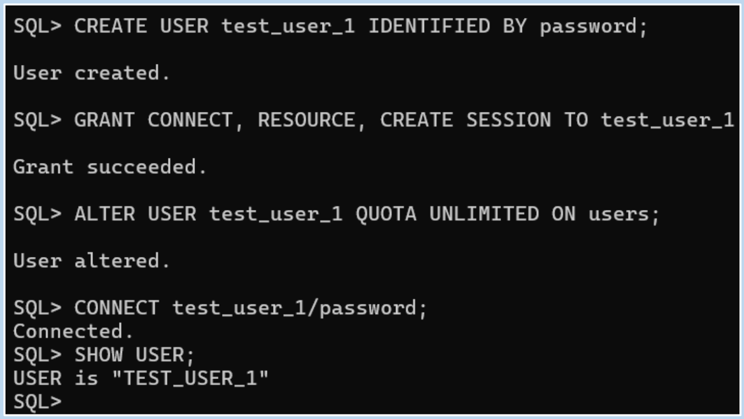
```
sh-4.4$ pwd
/opt/oracle/keystores/tde
sh-4.4$ ls
cwallet.sso  ewallet.p12  ewallet_2024112502504927.p12
sh-4.4$ pwd
/opt/oracle/keystores/tde
```

Creating a Table with TDE Column Encryption – For DBA

A new table will be created (‘*customer_table*’) with a new user schema. The *‘customer_table’* will have the column ‘*credit_card_number’* encrypted.

<u>Creating new user schema</u>

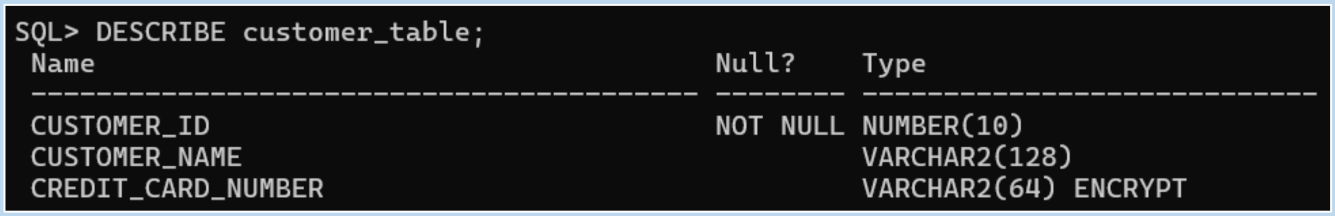
```
1.  CREATE USER test_user_1 IDENTIFIED BY password;
2.  GRANT CONNECT, RESOURCE, CREATE SESSION TO test_user_1;
3.  ALTER USER test_user_1 QUOTA UNLIMITED ON users;
4.  CONNECT test_user_1 /password;
5.  SHOW USER;
```



```
SQL> CREATE USER test_user_1 IDENTIFIED BY password;

User created.

SQL> GRANT CONNECT, RESOURCE, CREATE SESSION TO test_user_1

Grant succeeded.

SQL> ALTER USER test_user_1 QUOTA UNLIMITED ON users;

User altered.

SQL> CONNECT test_user_1/password;
Connected.
SQL> SHOW USER;
USER is "TEST_USER_1"
SQL>
```

<u>Creating table with encrypted column</u>

```
1.  CREATE TABLE customer_table (customer_id NUMBER(10) PRIMARY KEY,customer_name
    VARCHAR2(128),credit_card_number VARCHAR2(64) ENCRYPT);
2.  DESCRIBE customer_table;
```



```
SQL> DESCRIBE customer_table;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 CUSTOMER_ID                               NOT NULL NUMBER(10)
 CUSTOMER_NAME                                      VARCHAR2(128)
 CREDIT_CARD_NUMBER                                 VARCHAR2(64) ENCRYPT
```

Useful References

TDE Column Encryption Documentation (Oracle) - https://docs.oracle.com/en/database/oracle/oracle-database/21/asoag/encrypting-columns-tables2.html#GUID-A0E2C279-10F6-48D9-AB92-C547529C06EE

Key Stores, Key Vault types (Oracle) - https://docs.oracle.com/en/database/oracle/oracle-database/18/asoag/introduction-to-transparent-data-encryption.html#GUID-8178CBF5-78EB-4C6A-9AA5-8C05A9BEDC71

Setup Guide for Wallet Based TDE (Oracle) - https://docs.oracle.com/en/database/oracle/oracle-database/23/dbtde/get-started.html#GUID-FA569DEC-6FF3-4CA1-86AA-D27F29EDCA3C

Oracle TDE in AWS RDS (AWS) - https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Appendix.Oracle.Options.AdvSecurity.html

Securing Data at Rest on Amazon RDS Custom For Oracle with TDE (Part 1) - https://aws.amazon.com/blogs/database/secure-data-at-rest-on-amazon-rds-custom-for-oracle-with-tde-part-1-non-cdb-environments/

Securing Data at Rest on Amazon RDS Custom For Oracle with TDE (Part 2) - https://aws.amazon.com/blogs/database/secure-data-at-rest-on-amazon-rds-custom-for-oracle-with-tde-part-2-multi-tenant-environments/

Orace TDE with AWS CloudHSM (AWS) - https://docs.aws.amazon.com/cloudhsm/latest/userguide/oracle-tde.html