Fortanix®

Best Practices

FORTANIX SELF-DEFENDING KMS WITH ORCACLE TDE

Fortanix | 444 Castro St #305| Mountain View, CA 94041| ☎ +1 (628) 400-2043| ✉ info@fortanix.com | www.fortanix.com

TABLE OF CONTENTS

1.0 INTRODUCTION 2
1.1 Overview 2
2.0 BENEFITS OF USING TDE 2
3.0 WHO CAN CONFIGURE TDE? 3
4.0 SECURELY MANAGING TDE MASTER KEY 3
5.0 TYPE CONSIDERATION FOR TDE IMPLEMENTATION 4
5.1 When to Choose TDE Tablespace Encryption? 4
5.2 When to Choose TDE Column Encryption 4
6.0 WHAT TO EXPECT AFTER TDE IMPLEMENTATION? 5
7.0 WHAT TO DO TO ALLEVIATE THE TDE PERFORMANCE 5
8.0 TDE BEST PRACTICES 6
9.0 TDE BEST PRACTICE FOR FORTANIX SELF-DEFENDING KMS INTEGRATION 8
10.0 DOCUMENT INFORMATION 10
10.1 Document Updates 10
10.2 Document Updates 10

1.0 INTRODUCTION

This article describes the best practices for using Oracle Advanced Security Transparent Data Encryption (TDE).

1.1 OVERVIEW

- Oracle Advanced Security TDE provides the ability to encrypt sensitive application data on storage media completely transparent to the application itself. TDE addresses encryption requirements associated with privacy and security mandates such as PCI, HIPPA, and so on.
- TDE transparently encrypts data at rest in Oracle Databases. It stops unauthorized attempts from the operating system to access database data stored in files, without impacting how applications access the data using SQL. TDE can encrypt entire application tablespaces or specific sensitive columns. TDE is fully integrated with the Oracle database. Encrypted data remains encrypted in the database, whether it is in tablespace storage files, temporary tablespaces, undo tablespaces, or other files that Oracle Database relies on such as redo logs. Also, TDE can encrypt entire database backups (RMAN) and Data Pump exports.

2.0 BENEFITS OF USING TDE

- 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.
- 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.

- You can encrypt data with zero downtime on production systems by using online table redefinition or you can encrypt it offline during maintenance periods.
- 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.

3.0 WHO CAN CONFIGURE TDE?

- A database administrator or a database security admin can configure TDE. You must be granted the `ADMINISTER KEY MANAGEMENT` system privilege to configure Transparent Data Encryption (TDE).
- If you must open the Keystore at the mount stage, then you must be granted the `SYSKM` administrative privilege, which includes the `ADMINISTER KEY MANAGEMENT` system privilege and other necessary privileges.
- When you grant the `SYSKM` administrative privilege to a user, ensure that you create a password file for it so that the user can connect to the database as `SYSKM` using a password. This enables the user to perform actions such as querying the `V$DATABASE` view.
- To configure TDE column or tablespace encryption, you do not need the `SYSKM` or `ADMINISTER KEY MANAGEMENT` privileges. You must have the following additional privileges to create TDE policies on tables and tablespaces:
    - `CREATE TABLE`
    - `ALTER TABLE`
    - `CREATE TABLESPACE`

4.0 SECURELY MANAGING TDE MASTER KEY

Oracle TDE uses a 2-tier key architecture where the Master Encryption Key stored outside the database and the Data Encryption Key stored within the database is responsible for encryption and decryption of data that is stored inside a database. It is always a good practice to store the

Master Key in a Key Management Solution outside your database server to protect it from unintended deletion or server failures.

Fortanix Self-Defending KMS provides FIFS 140-2 Level 3 compliant hardware security module appliances to store and manage the TDE master keys and provides secure access to database end-points for TDE online access. By default, the Oracle TDE master key stored in Fortanix Self-Defending KMS does not expire unless you explicitly define the expiry.

5.0 TYPE CONSIDERATION FOR TDE IMPLEMENTATION

You can encrypt sensitive data at the column level or the tablespace level. At the column level, you can encrypt data using selected table columns. TDE tablespace encryption enables you to encrypt all 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.

It is often unclear for customers which type of TDE they should use that can provide better performance and suitable for their needs. This section will clarify that ambiguity and will provide a better understanding of the best practices while choosing TDE.

5.1 WHEN TO CHOOSE TDE TABLESPACE ENCRYPTION?

- It is useful if your table contains sensitive data in multiple columns.
- If you want to protect the entire table and not just individual columns.
- TDE tablespace encryption takes advantage of bulk encryption and caching to provide enhanced performance. It is useful when you are using an OLTP database that does bulk writing/updates.
- When you do not want to modify the table structure.

5.2 WHEN TO CHOOSE TDE COLUMN ENCRYPTION

- When you have fewer columns to encrypt
- Columns holding the sensitive information are pre-known.

6.0 WHAT TO EXPECT AFTER TDE IMPLEMENTATION?

- It is very important to take into consideration that the encrypt/decrypt operations are overhead to the existing plans. Encryption and decryption are typically CPU intensive operations and would always require additional CPU resources.
- The most common problem is caused by the fact that column encryption is unable to use the range scan on indexes, because when the index columns are encrypted (all or some of them) the index keys are not sorted in the same order as in the non-encrypted form. Consequently, with TDE column encryption, an index range scan becomes a full table scan.
- If using tablespace encryption, queries requesting a large number of encrypted blocks (especially full table scan) would suffer from performance degradations. It is the expected overhead when reading a full table from the disk into the buffer cache.
- RMAN Backup size may increase after the implementation of TDE. You can use Oracle advanced compression feature to reduce the size of the backup, optionally if you are using any third-party backup solution then you can use their deduplication and other mechanisms to adjust the compression ratio.
- There will be an additional storage overhead with TDE implementation.
    - In TDE Column encryption: between 1-52 bytes for each encrypted value caused by the following factor.
        - 20 bytes overhead for integrity check
        - 16 bytes overhead if 'SALT' is used to encrypt a column
        - Padding (AES uses 16 bytes, 3DES uses 8 bytes)
    - Tablespace encryption causes almost no storage overhead.

7.0 WHAT TO DO TO ALLEVIATE THE TDE PERFORMANCE

1. if the columns used in the predicates and joins of the statements are encrypted with SALT. Try encrypting these columns with NO SALT, as it generally should provide better performance.
2. For tablespace encryption and full table scan operations:
    - If a table is very large and queried mostly with full table scan operations, consider not storing it in an encrypted tablespace but use column encryption and encrypt just the sensitive columns.

- If a table is not very large, queried mostly with full table scan operations, and must reside in an encrypted tablespace, consider the possibility of keeping it as much as possible in the buffer cache by enabling the “keep buffer pool” and setting the table to use it. This way, only part of the blocks in the table would be read from disk and decrypted, significantly reducing the decryption overhead.
- Consider the possibility of building aggregate materialized views on such large tables residing in encrypted tablespaces. This way, the final data would reach in a faster way to the end user.
- The degree of parallelism, the number, and power of the CPU is also an important factor that can reduce the time taken for a full table scan. Consider increasing the degree of parallelism for these problematic tables.
- If the queries are not using aggregate functions and yet returning a small number of records, then indexes would likely help. The statements should be tuned, and indexes added as needed on the tables.

3. The NOMAC parameter enables you to skip the integrity check and also saves 20 bytes of disk space.

8.0 TDE BEST PRACTICES

The overhead of the TDE, be it column or tablespace encryption cannot be assessed without testing, it depends on the statement to statement. Please test thoroughly before implementing it.

Those are certain best practices that can be followed for a successful TDE implementation.

1. Periodic key rotation: Perform TDE master key rotation every 6 months or 1 year depending on your internal security compliance.

2. Full DB/table backup before TDE implementation: Take a full backup of the database or the table before encryption on tablespace or table column.

3. Safeguard master key: It is important to understand that if you lose the TDE master key, you will not be able to access the encrypted data. Hence it is extremely important to safeguard the TDE

master key from any unintended deletion or server failure. As per security compliance, it is always advisable to store the TDE master key outside the database server such as in an HSM.

4. Avoid full table scan: If your sensitive data is stored in an encrypted table column or in an encrypted tablespace, avoid full table scan as full tablespace scan will impact the performance by doing a lot of disks I/O and CPU cycles to decrypt the data.

5. Cache the small tables in buffer cache: It is always a better way to enhance the performance by keeping the small tables in the buffer cache for the encrypted tables.

6. Leverage hardware acceleration: Hardware acceleration can help to reduce encryption/decryption overhead, all modern Intel CPU's support AES-NI.

7. Use auto-login wallet: It is a good practice to use the Oracle DB auto-login wallet as it eliminates the manual effort of opening the Keystore from file or HSM during the database restart or other database/server maintenance activities.

8. TDE restrictions: TDE tablespace encryption encrypts/decrypts data during read/write operations, as opposed to TDE column encryption, which encrypts/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, are not applicable to TDE tablespace encryption.

The following list includes the restrictions that apply to TDE tablespace encryption:

- External Large Objects (BFILEs) cannot be encrypted using TDE tablespace encryption. This is because these files reside outside the database.
- You should use Oracle data dump for import and export, not the standard `exp` and `imp` utility.

9.0 TDE BEST PRACTICE FOR FORTANIX SELF-DEFENDING KMS INTEGRATION

As the database is a business-critical infrastructure in an enterprise, maintaining the high availability of a key management system that holds the database master key is absolutely important.

Fortanix Self-Defending KMS is designed to provide High availability to all critical infrastructure that needs access to the Fortanix Self-Defending KMS system for creating and accessing any security objects such as Oracle TDE master keys.

Here are some important suggestions that an Oracle DBA or security admin should consider when planning for Fortanix Self-Defending KMS integration with Oracle TDE.

- Appropriately configure the PKCS#11 library location. Oracle suggests keeping the PKCS#11 library in its designated location.

  For example: `/opt/oracle/extapi/{32,64}/hsm/<vendor_name>/<pkcs_lib_version>`
- You should create the `pkcs11.conf` file in `/etc/fortanix` directory. Make sure you should not change the file name `pkcs11.conf` to any other name.
- If you want to change the path of the `pkcs11.conf` file, you should export the patch for the following environment variable

  For example: `export FORTANIX_PKCS11_CONFIG_PATH=/PATH/pkcs11.conf`
- Make sure you keep the following entries in the `pkcs11.conf` file:
  - **`prevent_duplicate_opaque_objects = true`** (It will prevent duplicate object creation if it is already available in Fortanix Self-Defending KMS)
  - **`retry_timeout_millis = 60000`** (This will make sure PKCS will retry connecting Fortanix Self-Defending KMS before the session expires)
- If you are expecting a network latency in the communication channel between Fortanix Self-Defending KMS and Database Server, then you should consider increasing the fault tolerance of the database by setting the below as sysdba. (from 12.1 onwards)

```
ALTER SYSTEM SET "_heartbeat_period_multiplier"= xx SCOPE=SPFILE;
```

```
ALTER SYSTEM SET "_heartbeat_config"=AUTOCONNECT SCOPE=SPFILE;
```

Where `xx` is used in the following way to determine the heartbeat interval:

`xx + 3` (initial seconds) = `yyy` seconds.

For example, `2` minutes (`120` seconds), the `xx` value will be `39`.

`39x3 + 3` (initial seconds) = `120` seconds.

- If you are using 11gR2 (11.2.0.3 and 11.2.0.4) you should consider setting the following event which will improve resilience to minor network outage when using TDE with an HSM.

```
ALTER SYSTEM SET EVENT='28420 trace name context forever, level
   10:28421 trace name context forever, level 3' COMMENT='HSM
   heartbeat timeout and reconnect attempt' SCOPE=SPFILE;
```

- **Event 28420** is used to determine how many HSM heartbeats can fail before the wallet is closed. The HSM heartbeat fires every 3 seconds which means that a very short network outage can lead to wallet closure. For example, if event 28420 is set to level 5 then the RDBMS will allow 5 heartbeats to fail before closing the wallet, which would allow a 15 second loss of contact with the HSM before closing the wallet. This event should NOT be used to disable the heartbeat functionality. We strongly advise that this value is set no higher than 20 (i.e. one-minute loss of HSM connectivity before wallet closure).

- **Event 28421** will cause the HSM heartbeat to attempt to reconnect with the HSM once the wallet has been closed, and if successful, reopen the wallet. For example, if event 28421 is set to level 3 then every 3rd heartbeat will attempt to re-establish connection with the HSM. Event 28421 is only useful if HSM has been configured using the auto-open feature. This means that a local `cwallet.sso` maintains the HSM credentials in the secret store entry `ORACLE.TDE.HSM.AUTOLOGIN`.

10.0 DOCUMENT INFORMATION

10.1 DOCUMENT UPDATES

The latest published version of this document is located at the URL:

https://support.fortanix.com/hc/en-us/articles/360051875172-Fortanix-Self-Defending-KMS-with-Oracle-TDE-Best-Practices

10.2 DOCUMENT UPDATES

This document will typically be updated on a periodic review and update cycle.

For any urgent document updates, please send an email to: support@fortanix.com