An Oracle White Paper
February 2011

Oracle Advanced Security Transparent Data Encryption Best Practices

Introduction

This paper provides best practices for using Oracle Advanced Security Transparent Data Encryption (TDE). 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 public and private privacy and security mandates such as PCI and California SB1386. Oracle Advanced Security TDE column encryption was introduced in Oracle Database 10g Release 2, enabling encryption of application table columns, containing credit card or social security numbers. Oracle Advanced Security TDE tablespace encryption and support for hardware security modules (HSM) were introduced with Oracle Database 11gR1.

Important Concepts

- **Master encryption key** – The encryption key used to encrypt secondary keys used for column encryption and tablespace encryption. Master encryption keys are part of the Oracle Advanced Security two-tier key architecture.

- **Unified master key** – The unified master encryption key is generated with the first re-key operation in an Oracle Database 11g Release 2. The unified master key can be re-keyed (rotated), regardless if it is stored in the Oracle Wallet or a Hardware Security Module (HSM)

- **Table key** – Sometimes referred to as a column key, this key is used to encrypt one or more specific columns in a given table. Table keys were introduced in Oracle Database 10g Release 2. These keys are stored in the Oracle data dictionary, encrypted with the master encryption key.

- **Tablespace key** – The key used to encrypt a tablespace. These keys are encrypted using the master key and are stored in the tablespace header of the encrypted tablespace, as well as in the header of each operating system - file that belongs to the encrypted tablespace.

- **External Security Module –** A file or hardware device located outside the database that is used to store the master encryption key. In the context of TDE this could be the Oracle Wallet, a Hardware Security Module, or another key management system that supports the PKCS#11 interface.

- **Wallet** – A PKCS#12 formatted file outside of the database, encrypted based on password-based encryption as defined in PKCS#5

- **Hardware Security Module (HSM)** - A device used to secure keys and perform cryptographic operations. These devices can be standalone network based appliances or plug-able PCI cards. In the context of TDE, these devices can create and store the TDE master key.

- **Advanced Encryption Standard (AES)** – A symmetric cipher algorithm defined in the Federal Information Processing (FIPS) standard no. 197. AES provides 3 approved key lengths: 256, 192, and 128 bits.

- **PKCS#11** – A standard developed by RSA for communicating with cryptographic devices.

- **PKCS#12** – A file format standard published by RSA, used for storing cryptographic keys.

Hardware cryptographic acceleration with Intel® Xeon® 5600

Hardware cryptographic acceleration is available with AES-NI, a set of <u>N</u>ew <u>I</u>nstructions for the <u>A</u>dvanced <u>E</u>ncryption <u>S</u>tandard, in the Intel® Xeon® 5600 CPUs. Oracle Database 11gR2 Patchset 1 (11.2.0.2) TDE tablespace encryption automatically detects and leverages the hardware-based cryptographic acceleration for **decryption** of data; for hardware accelerated **encryption**, patch [10080579](#) is required. Hardware cryptographic acceleration for TDE column encryption is currently not supported.

Manage Transparent Data Encryption in Oracle Enterprise Manager

Most database maintenance and configuration tools are integrated in a convenient, easy-to-use Web interface called Enterprise Manager. Oracle Enterprise Manager **Database** Control is installed by default and used to manage a single, local database instance. Oracle Enterprise Manager **Grid** Control is an infrastructure that allows monitoring, configuring and maintaining many distributed databases from one central console.

The following command starts Oracle Enterprise Manager Database Control:

```
$ emctl start dbconsole
```

Point your Browser to `https://<hostname>:<port>/em` and provide user name and password of the user with sufficient privileges to manage a database, for example 'SYSTEM'.

On the main page of Oracle Enterprise Manager Database Control, click on the 'Server' tab, on the following page, click on 'Transparent Data Encryption' within the 'Security' group to reach the Transparent Data Encryption homepage.

Transparent Data Encryption and Oracle Database Vault

If your database is protected with Oracle Database Vault, separation of duties is enforced that includes controlling the authorizations of users in Enterprise Manager. In order to enable 'SYSTEM' or (any other database user with DBA privileges) to manage Transparent Data Encryption, this user has to be a 'Participant' or 'Owner' of the 'Data Dictionary Realm' in Oracle Database Vault. This change needs to be done by a user with the Database Vault 'owner' role.

Transparent Data Encryption Key Architecture

Encryption keys are the secrets used in combination with an encryption algorithm to encrypt data. Oracle Advanced Security TDE uses a two tier encryption key architecture, consisting of a master key and one or more table and/or tablespace keys. The table and tablespace keys are encrypted using the master key. The master key is stored in an external security module (ESM): The Oracle Wallet, Hardware Security Module (HSM), or external PKCS#11 compatible key management system.

Key Generation and Backup

If the TDE master key is stored in an Oracle wallet, it is generated by Oracle during the initial configuration of TDE. The master key is generated using a pseudo-random number generator inside the Oracle database. If an HSM device is used to store the master key, the HSM device itself creates it.

Always backup the wallet associated with the master key immediately after it is initially created, whenever the master key is changed, and before changing the wallet password. The wallet is a critical component and should be backed up in a secure location, on-site and offsite. If you are using an HSM device, follow the manufacturer's instructions for insuring the recoverability of keys in case the HSM fails.

Key Storage

The TDE master key is stored in an external security module: Either an Oracle wallet, HSM device, or external PKCS#11 compatible key management system. External security module support is dependent on your version of Oracle.

EXTERNAL SECURITY MODULE SUPPORT BY DATABASE VERSION			
DATABASE VERSION	MASTER KEY FOR …	… IN ORACLE WALLET	… IN HSM
Oracle Database 10gR2	TDE column encryption	Yes	No
Oracle Database 11gR1 (11.1.0.6)	TDE column encryption	Yes	Yes
	TDE tablespace encryption	Yes	No
Oracle Database 11gR1 (11.1.0.7)	TDE column encryption	Yes	Yes
	TDE tablespace encryption	Yes	Yes (no re-key)
Oracle Database 11g Release 2	TDE column and tablespace encryption (unified master key)	Yes	Yes

Key Exchange / Rotation

In TDE column encryption, both the master key and table keys can be individually re-keyed, providing for a granular implementation of various security policies. Re-keying of the master key does not impact performance or availability of your application, because it requires only decryption and re-encryption of the table keys and not the associated encrypted application data. Re-keying the table keys requires careful planning, since associated application data must first be decrypted and subsequently re-encrypted using the new table encryption key. Changing the table keys would be equivalent to performing a full table update. After upgrading to Oracle Database 11gR1, performing a TDE master key re-key operation will transparently create a separate TDE tablespace encryption master key in the Oracle Wallet. Tablespaces created using the ENCRYPT syntax will have any associated data files encrypted using the tablespace key stored in each tablespace header. The tablespace key itself will be encrypted using the new tablespace master key. After upgrading to Oracle Database 11g Release 2, performing a TDE master key re-key operation will either merge the two existing master keys to a unified master encryption key, or create a unified master encryption key. The unified master encryption key is used for both TDE column encryption and TDE tablespace encryption.

Note the restriction in Oracle Database 11gR1 that tablespace master keys and tablespace keys cannot be re-keyed. If a re-key is required for a given encrypted tablespace, Oracle recommends moving the data to a new encrypted tablespace. Please see the section on tablespace re-key restrictions on page 9 for recommendations on moving data to a new tablespace to achieve a re-key operation. Oracle Database 11g Release 2 allows the rotation of the unified master encryption key.

RE-KEY SUPPORT				
	TDE COLUMN ENCRYPTION		TDE TABLESPACE ENCRYPTION	
	Master Key	Table Keys	Master Key	Tablespace Keys
Oracle Database 10gR2 and 11gR1	Yes	Yes	No	No
Oracle Database 11g Release 2	Yes (*)	Yes	Yes (*)	No

(*): Unified master encryption key for TDE column encryption and TDE tablespace encryption

Oracle Wallet: Directory and File Permissions

When using the Oracle Wallet, Oracle recommends restricting the associated file and directory permissions. In addition, a strong password should be used when setting up the wallet. The Oracle Wallet is the default external security module used to store the (unified) TDE master encryption key. Oracle recommends placing the Oracle Wallet outside of the $ORACLE_BASE directory tree to avoid accidentally storing the wallet with the encrypted data on a backup tape, for example:

```
/etc/ORACLE/WALLETS/<Oracle software owner user name>
```

Since /etc is owned by 'root', these directories are created by 'root'; when done, change ownership to 'oracle:oinstall' and set the permissions to 'oracle' only:

```
# cd /etc
# mkdir -pv ORACLE/WALLETS/oracle
# chown -R oracle:oinstall ORACLE
# chmod -R 700 ORACLE
```

Set the ENCRYPTION_WALLET_LOCATION parameter in sqlnet.ora to the newly created directory:

```
ENCRYPTION_WALLET_LOCATION =
  (SOURCE = (METHOD = FILE)
    (METHOD_DATA =
      (DIRECTORY = /etc/ORACLE/WALLETS/oracle/)))
```

Initialize the wallet and add the master encryption key using Enterprise Manager or the SQL*Plus command line interface:

```
SQL> alter system set encryption key identified by "password";
```

After successful creation of the wallet and master key, reduce permissions on the wallet file from the initial value, determined by 'umask' for the 'oracle' user, to:

```
$ cd /etc/ORACLE/WALLETS/oracle
$ chmod 600 ewallet.p12
```

It is highly recommended to always backup the wallet at the same time when backing up your database, but do not include the wallet on the same media as the database backup. Also, backup the wallet before any manipulation of its content, whether performing a master key re-key operation, or changing the wallet password.

(Local) Auto-Open vs. Encryption Wallet

The encrypted wallet ('ewallet.p12') offers strong protection of the master key, by encrypting the wallet with the wallet password, following the PKCS#5 standard for password-based encryption. Opening the wallet is a manual operation and must be performed to make the master encryption key available to the database. Optionally, the master key can be copied into an 'auto-open' wallet. This can be done either using Oracle Wallet Manager or the 'orapki' utility:

```
$ orapki wallet create -wallet <wallet_location> -auto_login
```

This command creates an auto-open wallet ('cwallet.sso'). In order to significantly strengthen your security when using an auto-open wallet, a **local** auto-open wallet can be created,

starting with Oracle Database 11.1.0.7; it does not open on any machine other than the one it was created on:

```
$ orapki wallet create –wallet <wallet_location> -auto_login_local
```

Important - Do not delete the original encryption wallet. Re-keying the master key requires the original encryption wallet to be present. When the master key is re-keyed, the corresponding (local) auto-open wallet is updated automatically.

Backup the auto-open wallet in a separate location from the encrypted data. Storing the auto-open wallet with the encrypted data provides no security, since the wallet and data on a stolen or misplaced tape or disk would have no protection.

Strong Wallet Password

The wallet password is critical to providing strong security. If the wallet password is compromised, someone with access to the operating system could simply copy the database files and wallet and use the wallet password to make the master key available to a database and decrypt the encrypted application data. It is easy to see that the password needs to be strong, yet easy to remember, since a forgotten wallet password cannot be recovered. One way to come up with a strong yet easy to remember password is to take the first characters of each word in an easy-to-remember sentence: "I work from 9 to 5 almost every day of the week" would give "`Iwf9t5aedotw`", which satisfies the common requirements for good passwords: It contains numbers as well as upper- and lower case characters, and it is longer than the recommended minimum of 10 characters. The sentence is very easy to remember, but you don't have to remember the complex password itself at all.

Split knowledge about the Wallet password

When Enterprise Manager Database Control in Oracle Database 11g is used, the wallet password is always masked, so it is not only easy to hide from the DBA, but can also be split easily between different custodians: Person A enters the first part of the password before Person B enters the 2nd half of the password, without Person B being able to see what Person A typed into the password field.

In Oracle Database 10gR2, where the wallet password on the SQL*Plus command line is displayed in the clear, password splitting is not possible. For customers to translate the need for 'split knowledge about the encryption key' to 'split knowledge about the Wallet password', the following script provides a possible work-around:

Create a user '`Sentinel`' in your database with only '`create session`' and '`alter system`' privileges

Create a Secure External Password Store (SEPS), following the instructions in this document. As explained in this document, create an entry in `tnsnames.ora` called '`keyholder`'; confirm with '`$ tnsping keyholder`' that the entry is correct. Add credentials for the user '`sentinel`' to

the SEPS: `$ mkstore -wrl . -createCredential keyholder sentinel <password>`
Try to connect to the database with '`$ sqlplus /@keyholder`'

This script ('`set_key.sh`') creates a new wallet in the defined location (if it does not exist) and adds a new TDE master encryption key to a wallet, or re-keys the master encryption key:

```
#!/bin/bash
#
get_pwd1(){read -s -p "1st half of password: " pwd1}
get_pwd2(){read -s -p "2nd half of password: " pwd2}
set_key(){sqlplus /@keyholder @set_key.sql $pwd1$pwd2}

get_pwd1
get_pwd2
set_key
```

The SQL script '`set_key.sql`':

```
set termout off;
alter system set encryption key identified by "&1";
set termout on;
exit
```

A similar script can be written to open the wallet, or to change the wallet password using the '`orapki`' command line tool in 11.1.0.7 or later (see next paragraph).

Oracle recommends performing the wallet creation and master key initialization on the database server itself. If you plan to use a remote machine to perform the operation, enable network encryption between the client and database server so that the communication between both machines is secure.

Changing the Wallet Password

Before changing the password on an existing wallet, be sure you have backed up the existing wallet. After changing the password, verify that you can open the wallet using the new password.

Changing the password is independent from changing the master key. A pseudo-random number generator generates the master key, while the wallet password is used as the key to encrypt the wallet. Prior to Oracle Database release 11.1.0.7, changing the wallet password required using Oracle Wallet Manager. Before changing the password of an existing wallet, be sure you have backed up the wallet. After changing the password, verify that you can open the wallet using the new password. If you can't open the wallet with the new password, simply restore the backup copy of the wallet and try changing the password again. Starting with Oracle Database release 11.1.0.7, the '`orapki`' utility has been enhanced to enable wallet password changes from the command line:

```
$ orapki wallet change_pwd -wallet <wallet_location>
```

Using Hardware Security Modules (HSM)

The master key for TDE column encryption and (from 11.1.0.7) TDE tablespace encryption can be generated and stored in a Hardware Security Module (HSM). Because master keys never leave the HSM device in clear text, it cannot be loaded into database memory; therefore, Oracle sends the encrypted table or tablespace keys to the HSM device, where they are decrypted and returned to the database to process encrypted data. Table keys are not cached in database memory; for each new access to encrypted application table columns that starts after a previous access to the same encrypted column has ended, the table keys are decrypted by the master encryption key in the HSM. Tablespace keys are loaded into database memory when first used and are cached there until the database is shutdown.

HSM support for creating and storing the TDE column encryption master key requires Oracle Database version 11.1.0.6 or higher. HSM support for creating and storing (but not rotating/re-keying) the TDE tablespace encryption master key requires Oracle Database version 11.1.0.7. Starting with Oracle Database 11g Release 2, the **unified** master encryption key for both TDE tablespace encryption and TDE column encryption can be re-keyed, regardless of its storage location (Oracle Wallet or HSM).

The database communicates with the HSM device via the open industry standard protocol `PKCS#11`. Leading HSM vendors have certified their appliances with TDE. The software delivered by the HSM vendor typically includes a `PKCS#11` library and associated configuration files. Please follow the vendor specific installation procedures and documentation. Verify that either the `PKCS#11` library or, on Unix and Linux systems, a symbolic link to the library, is stored in this pre-defined directory:

```
/opt/oracle/extapi/32|64/hsm/<vendor>/<version>/libname.ext
```

where `<vendor>` is the name of your HSM vendor, `<version>` is the version of the `PKCS#11` library; the filename of the library itself has to begin with '`lib`'.

If you already used TDE column encryption with a software wallet and you have encrypted columns in your application tables, the existing master key from the wallet can be migrated to the HSM device. In this case, your '`sqlnet.ora`' file needs to contain this entry:

```
ENCRYPTION_WALLET_LOCATION=
   (SOURCE = (METHOD = HSM)
     (METHOD_DATA =
       (DIRECTORY = /etc/ORACLE/WALLETS/oracle)))
```

Migrating the TDE column encryption master key (or the unified master encryption key in Oracle Database 11g Release 2) to an HSM:

```
SQL> alter system set encryption key identified by
"HSM_authentication_string" migrate using "wallet_password";
```

With this command, the existing table keys are decrypted with the master key in the software wallet, and re-encrypted with the new master key in the HSM device.
In Oracle Database 11g Release 2, table keys for TDE column encryption and tablespace keys for TDE tablespace encryption are decrypted with their individual master keys from the wallet, and re-encrypted with a unified master encryption newly generated in the HSM.

If you have data protected by a wallet-based master key, you need to retain the original software wallet. This will be the case if …

- TDE has been actively used in your database, so that data in redo and log files is encrypted under the wallet-based master key,

- you have created encrypted backups that may need to be used in the future,

- or you used the TDE master key for creating an encrypted Oracle Data Pump export.

After migrating the master encryption key from an Oracle Wallet to an HSM, it is mandatory to change the Oracle Wallet in one of two ways:

- Change the wallet password to match the "HSM_authentication_string".

- If changing the wallet password is not feasible:

  - create a (local) auto-open wallet from the encryption wallet, **and**

  - move the encryption wallet out of the ENCRYPTION_WALLET_LOCATION specified in sqlnet.ora to a secure location, but do **neither** delete the encryption wallet, **nor** forget the wallet password. The Oracle Wallet may contain master encryption keys that were used to encrypt RMAN backup files or Oracle Data Pump export files.

In Oracle Database 11.1.0.7 the master key for TDE tablespace encryption can be created and stored on an HSM device. This capability exists only if you have not performed a re-key operation or created an encrypted tablespace using the software wallet. This restriction is due to the fact that tablespace keys cannot be re-keyed in Oracle Database 11gR1. Migrating to an HSM device is a re-key operation. To migrate the TDE column encryption master key to the HSM device use the same command as shown above.

It is highly recommended to install patch 9453959 when using TDE in Oracle 11.1.0.7. This patch corrects bugs 8211698, 7563307, 9034189, and 7563237, as well as 8909973, which allows TDE in 11.1.0.7 address individual HSM partitions.

All TDE commands (open and close the wallet, master key generation, re-key of master and table keys) are the same as with local software wallet, but key management takes place on the HSM device.

Most HSM vendors allow partitioning of their devices; with Oracle Database 11g Release 2 and patch 9229896, these named slots can be used for TDE key management. To migrate from Oracle Wallet to an HSM partition, use the following syntax:

```
SQL> alter system set encryption key identified by
"HSM_authentication_string|<slot_name>" migrate using "wallet_password";
```

TDE Tablespace Encryption

An Oracle database consists of at least two logical storage units called tablespaces, which collectively store all of the database's data. Each tablespace in an Oracle database consists of one or more files called datafiles, which are physical structures that conform to the operating system in which Oracle Database is running. Nearly all databases have several additional tablespaces to store application specific data.

With Oracle Database 11g, new tablespaces can be defined as encrypted. Defining a tablespace as encrypted means the physical data files created on the operating system will be encrypted. Any tables, indexes and other objects defined in the new tablespace will be encrypted by default with no additional storage space requirements. During data reads, the Oracle database will automatically decrypt data before it arrives in database memory (SGA). Data that is moved out of the SGA and written to the file system will be encrypted. TDE tablespace encryption provides optimal performance by enabling existing indexes and foreign keys to continue working as they were before encryption was turned on. Execution plans remain the same and the requirement to identify individual columns to encrypt is completely eliminated.

Oracle Applications certified for TDE Tablespace Encryption

The following Oracle applications have been certified with TDE tablespace encryption in Oracle Database 11.1.0.7 and Oracle Database 11g Release 2:

- Oracle E-Business Suite (see http://blogs.oracle.com/stevenChan/certifications.html for current updates)

- Oracle PeopleSoft Enterprise 8.48 and later (Migration script and detailed implementation guide)

- Oracle Siebel CRM 8.0 and later

- Oracle JD Edwards EnterpriseOne

- SAP 6.40_EX2 and later (Oracle Database 11g Release 2 only, SAP note 974876)

Internal benchmark tests and customers reported a performance impact of 4 to 8% in end-user response time, and an increase of 1 to 5% in CPU usage.

Moving Your Application Data to Encrypted Tablespaces

Oracle Database 11g supports encrypting new tablespaces only. Application data can be migrated from existing, un-encrypted tablespaces to new encrypted tablespaces using these steps:

- Backup the database using your standard backup procedures

- Export all application tablespaces with Oracle Data Pump Export ('`expdp`'), optionally compressing the dump file for faster processing and reduced storage

- Create encrypted versions of the clear-text tablespaces:

  - Using '`dbms_metadata.get_ddl`', extract the original DDL (data definition language) used to create the application tablespaces, and spool them to a SQL script

  - Append '`ENCRYPTION [using '<algorithm>'] DEFAULT STORAGE(ENCRYPT)`' to each '`CREATE TABLESPACE`' command, without changing any of the other parameters.

  - Drop the original unencrypted application tablespaces, either with:
    `SQL> drop tablespace <name> including contents and datafiles;` or:
    `SQL> drop tablespace <name> including contents` **`keep`** `datafiles;` if you want to use operating system level commands like '`sdelete`' or '`shred`' to securely delete the old clear text datafile.

  - Create the encrypted tablespaces by running the edited script

- Import all application tablespaces with Oracle Data Pump Import ('`impdp`')

- Verify application is working properly

This procedure requires downtime, which is not always feasible. To migrate to encrypted tablespaces while the application remains fully available, Oracle recommends using Online Table Redefinition, a mature high-availability feature of the Oracle Enterprise Edition. A ready-to-run script, complemented with a detailed implementation guide, is available. The script can be easily modified to reflect your own migration needs.

Tablespace Re-Key Restrictions in Oracle Database 11gR1

Oracle Database 11gR1 does not support the re-key of the TDE tablespace master key or associated tablespaces keys. If it becomes necessary to change the master key associated with encrypted tablespaces, a slightly modified version of the script mentioned before can be applied to individual or all application tablespaces. Alternatively, use the Data Pump utility to extract the application data from the encrypted tablespaces, create new encrypted tablespaces, and then import the data into the new encrypted tablespace using Oracle Data Pump Import (`impdp`).

- Backup the database using your standard backup procedures.

- Export the application tablespaces with Oracle Data Pump '`expdp`', optionally compressing, and encrypting the dump file with a password (do not use the current master key to encrypt the dump file).

- Extract the DDL used to build the encrypted tablespaces (using '`dbms_metadata.get_ddl`' and spool to a SQL file.

- Drop the original encrypted application tablespaces, '`including contents and datafiles`'.

- Build new encrypted tablespaces using the script created in step 2., which are now encrypted with the new tablespace key.

- Import the application tablespaces with Oracle Data Pump '`impdp`'.

- Verify application is working properly.

TDE Column Encryption

TDE column encryption transparently encrypts sensitive data written to application table columns. This can be accomplished by marking sensitive columns as 'encrypted' in Enterprise Manager Database Control, or by appending the '`encrypt`' key word to the SQL DDL statement. Existing data types remain the same so the encryption is completely transparent to the existing application. Each table with one or more encrypted columns has its own table encryption key; these are stored in the data dictionary, encrypted with the master key.

When data is written to an encrypted column, sensitive values are encrypted immediately before they are written to disk. When an authorized user selects data from the database, the data is automatically decrypted and presented in clear text. As with TDE tablespace encryption, TDE column encryption protects against direct access to media by privileged operating system users as well as lost or misplaced tapes and disk drives.

To increase performance when encrypted data is processed, each table has its own table key that is used for all encrypted columns in that specific table. These table keys are decrypted with the master key prior to processing encrypted data, and stay decrypted for the duration of the transaction.

Oracle Application certified for TDE Column Encryption

The following Oracle applications have been certified with TDE column encryption in Oracle Database 10gR2 and 11g (10.2.0.4/5, 11.1.0.7 or 11.2.0.1 are recommended):

- Oracle E-Business Suite (see http://blogs.oracle.com/stevenChan/certifications.html for current updates)

- Oracle PeopleSoft Enterprise 8.46 and later

- Oracle Siebel CRM 7.7+

- SAP 6.40 and later (SAP note 974876)

- iFlex FlexCube 10.0

- RETEK Retail Sales Audit (RESA):

    o   RESA 12.0 and 13.0 (in Oracle Database 10gR2)

    o   RESA 13.1 (in Oracle Database 11gR1)

- Oracle Internet Directory 10.1.4.2

Identifying Sensitive Columns

Identifying tables and columns that contain sensitive data such as social security numbers and credit cards can be difficult, especially in large applications. One technique that can be useful is to search the Oracle data dictionary for column names and data types that are frequently used to store such information. Here's an example of using SQL to identify columns that might contain social security numbers:

```
SQL> select column_name, table_name, data_type from user_tab_cols where
column_name like '%SSN%' or
column_name like '%SECNUM%' or
column_name like '%SOCIAL%';
```

The user executing this statement is the primary user or schema that owns the application tables. This technique can be used for other types of information as well by simply substituting another string such as "PIN" in the SQL text.

Encrypting indexed columns

An indexed column can be encrypted if the column is used for equality searches and the index type is a B-tree index (normal, not DESCending). An index over one or more encrypted columns can be built only when these columns are encrypted using the 'no salt' syntax. If encrypted columns are used to build an index, the corresponding columns in the index are encrypted as well. Before the SQL statement is processed, the value in the SQL text that targets the encrypted column is encrypted with the table key of the target table; the database checks the index, matches the encrypted value, finds the rowid in the index, and presents the corresponding row from the base table. Following this procedure, the performance impact for equality searches can be substantially minimized. Note that range scans ('between' clauses) cannot use the encrypted index. However, personally identifiable information (PII) is rarely used in range scan operations. The encryption algorithm (default AES192) is defined per table, even if the statement can be given at any column definition in the 'create table' statement.

Reducing the storage overhead

TDE column encryption differs from TDE tablespace encryption in the area of storage requirements; after encryption, an encrypted value can be between 1 and 52 bytes longer than the clear text value. By default, TDE column encryption adds 'salt' and an integrity check (Message Authentication Code, MAC) to each encrypted value. Here we explore when you can forgo those security features to save storage overhead.

Basic encryption is deterministic, which means that a given plaintext will always encrypt to the same cipher text. This property is less secure when the uniqueness of the values in a column is not given. For example, when a column contains sensitive patient information about a rare medical condition, most patients would enter 'No', while only a few would enter 'Yes'. All of the cipher texts for 'Yes' would be identical, as would be those for the negative answers. Even though their responses are encrypted, people that suffer from that rare condition would be easy to identify. To get beyond this limitation, each clear text value is modified with 'salt', a random, 16-byte string. The resulting output using identical plaintext inputs generates completely different cipher texts. However, if you can guarantee that all values in a column are unique (for example when a 'UNIQUE INDEX' has been applied to the column), you can set the 'NO SALT' parameter when encrypting a column to reduce storage by 16 bytes for each cell. Salt is defined per column; one table can contain columns that are encrypted with or without salt at the same time.

For further protection, a 20-byte integrity check is appended to each encrypted value to provide tamper detection for the cipher text. Starting with Oracle Databases 10.2.0.4 and 11.1.0.7, the 'NOMAC' parameter can be used with TDE column encryption to omit the generation and storage of these additional 20 bytes. 'NOMAC' is defined on a table level; even though 'NOMAC' can be specified on one or more columns, it applies to all encrypted columns in a table.

The final type of storage overhead associated with column-level encryption is unavoidable. Each plaintext value is padded out to the next 16 byte if encrypted using AES; it's padded out to the next 8 byte when encrypted using 3DES168: If a clear text value requires 9 byte of storage, it is expanded to 16 bytes; if the clear text value requires 16 byte, it is expanded to 32 bytes (24 for 3DES168), and so on.
It is highly recommended to install patch 7639262 for TDE column encryption in 10.2.0.4 and patch 8421211 for TDE column encryption in 11.1.0.7 to greatly improve performance for a certain type of queries and to correct the behavior of TDE column encryption when applied to a column which is part of a composite index, where other columns than the encryption candidate are used for functional indexes. Please contact Oracle Support if a patch is not available for your version/platform combination.

Encrypting Columns in Gigabyte and Terabyte Tables

Sometimes, tables have so many rows and the application downtime window is so small, that the 'UPDATE' operation necessary to encrypt one or more columns would take too long, even though 'READ' access to the table is still possible while it's being updated.

Often, these tables that are the foundation of your business, containing Personally Identifiable Information (PII) that needs to be protected through encryption, but they are constantly updated, and cannot be taken down without interrupting your business.

For those tables, Oracle provides Online Table Redefinition, offering a transparent method to change table characteristics while the source table is fully available. For detailed steps, consult the documentation, but here is a short form:

- Create an interim table that has the desired characteristics you want the source table to have after the procedure, for example: One column is encrypted, while all other columns remain unchanged.

- Start the redefinition process while the source table is fully available.

- After the final step (where the source table is offline for a short moment; transparent to users and applications, with no data loss!), delete the interim table.

Disabling and re-enabling Transparent Data Encryption

Usually, when a wallet is deleted, whether or not data has been encrypted, and a new wallet and master encryption key are to be created, the error message 'ORA-28374: typed master key not found in wallet' is displayed.

The following procedure **cannot** be used to recover or replace a lost wallet, wallet password, or TDE master encryption key; it's provided for administrators who at one point in time created a wallet with TDE master encryption key, never encrypted data, and decided to not use TDE and deleted the wallet, and then decided to eventually use TDE.

After installing patch 8682102, perform log switches to cycle through all redo logs, then create a new wallet and TDE master encryption key.

TDE Tablespace Encryption or TDE Column Encryption?

In Oracle Database 11gR1, security administrators or DBAs can choose between TDE tablespace encryption and TDE column encryption; here are some guidelines:

TDE TABLESPACE ENCRYPTION OR TDE COLUMN ENCRYPTION?

CHOOSE TDE COLUMN ENCRYPTION IF …:	CHOOSE TDE TABLESPACE ENCRYPTION IF …:
Location of sensitive information is known	Location of sensitive information is unknown
Less than 5% of all application columns are encryption candidates.	Most of the application data is deemed sensitive, or **multiple** national and international security and privacy mandates apply to your industry
Data type and length is supported by TDE column encryption	Not all data types that hold sensitive information are

	supported by TDE column encryption
Encryption candidates are not foreign-key columns	Encryption candidates are foreign key columns
Indexes over encryption candidates are normal B-tree indexes	Indexes of encryption candidates are functional indexes
Application does not perform range scans over encrypted data	Application searches for ranges of sensitive data
Increase in storage by 1 to 52 bytes per encrypted value	No storage increase acceptable
Performance impact depends on percentage of encrypted columns; how often the encrypted values are selected or updated, the size of encrypted data, and other variables.	Constant performance impact below 10%

Clear-Text Copies of Encrypted Data

During the lifetime of a table, data may become fragmented, re-arranged, sorted, copied and moved within the tablespace; this leaves 'ghost copies' of your data within the database file. When encrypting an existing column, only the most recent 'valid' copy is encrypted, leaving behind older clear-text versions in ghost copies. If the data file holding the tablespace is directly accessed, bypassing the access controls of the database (for example with an hex - editor), old clear text values might be visible for some time, until those blocks are overwritten by the database. To minimize this risk, please follow these recommendations:

- Backup the database using your standard backup procedures.

- Create a new tablespace in a new data file (`CREATE TABLESPACE ...  `)

- Encrypt sensitive data in the original tables (`ALTER TABLE ...  ENCRYPT`)

- Move all tables (with or without encrypted columns) from the original tablespace into the new data file (`ALTER TABLE ...  MOVE ...  `)

- Verify the application is working properly.

- Drop the original tablespace (`DROP TABLESPACE`). Do not use the 'and datafiles' parameter; Oracle recommends to use stronger methods for OS – level operations.

- Use 'shred' or other commands for your platform to delete the old data file on the OS level.

The last step is recommended to lower the probability of being able to find ghost copies of the database file, generated by either the operating system, or storage firmware.

Attestation

In order to present proof of encryption, for example upon an auditor's request, Oracle provides views that document the encryption status of your database. For TDE column encryption,

please use the view 'dba_encrypted_columns', which lists the owner, table name, column name, encryption algorithm, and salt, for all encrypted columns. For TDE tablespace encryption, the following SQL statement lists all encrypted tablespaces with their encryption algorithm and corresponding, encrypted, data files:

```
SQL> SELECT t.name "TSName", e.encryptionalg "Algorithm", d.file_name
"File Name" FROM
v$tablespace t, v$encrypted_tablespaces e, dba_data_files d WHERE
t.ts# = e.ts# and t.name = d.tablespace_name;
```

The next SQL statement lists the table owner, tables within encrypted tablespaces, and the encryption algorithm:

```
SQL> SELECT a.owner "Owner", a.table_name "Table Name", e.encryptionalg
"Algorithm", FROM
dba_tables a, v$encrypted_tablespaces e WHERE
a.tablespace_name in (select t.name from v$tablespace t,
v$encrypted_tablespaces e where t.ts# = e.ts#);
```

Oracle Data Guard

Physical Standby

Oracle Data Guard Physical Standby works with Transparent Data Encryption beginning with the first release of Oracle Database 10g Release 2. The encrypted application data stays encrypted while redo log files are transferred from the primary to the secondary databases. However, the master key from the primary database needs to be present on the secondary site only when the secondary site is in READ ONLY mode or after a failover, but not for applying the redo logs. It is recommended to copy the primary wallet over to the secondary sites, so that in a case of a failover, all data is quickly available. In addition, the Oracle Wallet can optionally be converted to an auto-open wallet, making the master key available to the secondary database automatically when the database is brought online. Encrypted data remains encrypted in log files and during transit when the log files are shipped to the secondary database; Oracle Network Encryption is optional.

When the master key on the primary database is re-keyed, the wallet needs to be recopied over to all secondary sites. If the wallet is open on the secondary site it would need to be closed (which removes the old master key from memory), the new wallet could then be opened so that the new master key is loaded into database memory, where it is stored obfuscated.

Customers concerned about the wallet being used to access sensitive data on the standby site can change the wallet password on the secondary site using Oracle Wallet Manager. However, this can lead to increased password management complexity and the possibility of delayed availability of the standby site should the password be forgotten.

Logical Standby

Oracle Data Guard Logical Standby works with Transparent Data Encryption beginning with Oracle Database 11gR1. The master key needs to be present and open at the secondary site for SQL Apply to decrypt the data that it reads from the encrypted log files. The same master key is also used to optionally encrypt the incoming data while it is written to the Logical Standby database.

Oracle Streams

Oracle Streams works with TDE starting in Oracle Database 11g. Encrypted data is decrypted by the Streams engine to allow data transformation (character sets, database versions, platforms, etc.) and is not encrypted while being transmitted to the other databases, hence, encrypting the traffic with Oracle Advanced Security Network Encryption is recommended. When the receiving side cannot be reached and data needs to be stored temporarily, data originally encrypted is stored encrypted on disk. Prior to Oracle Database 11g Oracle streams treats encrypted columns as 'unsupported data types' and skips the associated tables. For the receiving databases, the local wallet does not need to be identical to the source wallet and master key, since the sensitive content arrives in clear text.

Active Data Guard

During a master key re-key operation on the primary, it generates a redo marker with a new master key ID which will be shipped to the standby databases. The standby will then perform the same master key re-key operations with the master key ID, given it can also access the HSM.

Oracle Transparent Data Encryption and Oracle GoldenGate

ORACLE TRANSPARENT DATA ENCRYPTION AND ORACLE GOLDEN GATE

ORACLE GOLDEN GATE VERSION	TDE COLUMN ENCRYPTION	TDE TABLESPACE ENCRYPTION
Current release	Partial support for all DB versions if encrypted columns are: • CHAR and VARCHAR2 data types, and • not primary key for table	Not supported
11.1.1.1 (ETA: tbd)	Full support in Oracle Database 10.2.0.5 and 11.2.0.2	

Oracle Transparent Data Encryption and Oracle RMAN

With a simple command, RMAN backups to disk can be encrypted **and** compressed:

```
RMAN> connect target <ORACLE_SID>/<SYS password>;
RMAN> set encryption on;
RMAN> backup as compressed backupset database;
```

ORACLE RMAN ENCRYPTION AND COMPRESSION OF DATA ENCRYPTED BY TDE

DATA	BACKUP WITH …		
	COMPRESSION	ENCRYPTION	COMPRESSION AND ENCRYPTION
NOT ENCRYPTED	Data compressed	Data encrypted	Data compressed first, then encrypted
ENCRYPTED WITH TDE COLUMN ENCRYPTION	Data compressed; encrypted columns are treated as if they were not encrypted, resulting in low compression ratio for encrypted columns.	Data encrypted; double encryption of encrypted columns.	Data compressed first, then encrypted; encrypted columns are treated as if they were not encrypted; double encryption of encrypted columns
DATA ENCRYPTED WITH TDE TABLESPACE ENCRYPTION	Encrypted tablespaces are decrypted, compressed, and re-encrypted; Clear text tablespaces are compressed and encrypted (after compression, they cannot be distinguished from encrypted data).	Encrypted blocks are passed through to the backup unchanged; clear text blocks are encrypted for backup.	Encrypted blocks are decrypted, compressed, and re-encrypted; Clear text blocks are compressed and encrypted.

Oracle RMAN can compress and encrypt backups to disk. Either the master key from the source database can be used, or, if the data is to be recovered into another database, the encryption key can be generated from a password, which eliminates the need to share the master encryption key. It also allows a 'dual' mode, where the backup file can be decrypted either by making the correct master key available, or by providing the correct password.

Real Application Clusters (RAC)

Oracle Wallet management in Oracle RAC

When Oracle Advanced Security TDE is configured to use an Oracle Wallet for the master key, there is a 1:1 relationship between the wallet/master key and database; this is also true for Real Application Clusters (RAC) configurations in Oracle Database 11gR1. Once TDE is enabled on the first instance, the wallet and the local sqlnet.ora file need to be copied to all other instances and manually opened for the master key to be loaded into each instance's memory.

Likewise, when the master encryption key is rekeyed on one instance, the wallet needs to be copied to all other instances in this cluster; close the wallet to remove the old master key from memory and open it again to load the new master key. Oracle does **not** support storing the Oracle Wallet on a shared drive, since the wallet may become corrupted when one instance re-keys the master key without properly updating the other instances.

In Oracle Database 11g Release 2, Oracle recommends to store the Oracle Wallet in a centralized location: Create an ACFS file system in ASM using 'asmca' (ASM Configuration Assistant) and store the wallet there; the entry in sqlnet.ora in all instances looks like this:

```
ENCRYPTION_WALLET_LOCATION=
  (SOURCE = (METHOD = FILE)
    (METHOD_DATA =
      (DIRECTORY = /opt/oracle/acfsmounts/data_tdevolume/wallet_dir/)))
```

This file system is mounted automatically when the instances start. Opening and closing the wallet, as well as commands to set or rekey/rotate the TDE master encryption key are synchronized between all nodes.

It is recommended to enable TDE with a shared wallet **during a planned downtime**; if the changes to the sqlnet.ora file are applied on all instances of a **running** RAC system, restarting the individual instances one by one (except the first one where the wallet was created) is necessary to initially synchronize the wallet status and to update the view gv$encryption_wallet with the DIRECTORY entry from sqlnet.ora.

If the Oracle Wallet cannot be stored on an ACFS file system, it needs to be copied to all instances: First create wallet and master key on the first instance, and then copy the wallet to all other instances. Wallet operations that do not update the wallet content (wallet open/close commands) **are** synchronized between RAC instances, even if the wallet is not stored in a central location, while master key set key or rekey operations are only synchronized with a centralized, shared wallet.

To use an HSM device in your RAC environment, first install and test the HSM vendor's software (PKCS#11 library and configuration file) in each node. Then, follow the instructions on configuring hardware security modules in this document.

Protecting the Oracle Wallet with ACFS access controls

When the TDE Wallet is stored in an ACFS file system, additional access protection and separation of duty can be implemented using the ACFS Security feature starting with Oracle Database 11gR2 11.2.0.2 on Linux. To use the ACFS Security feature, it must first be initialized for the cluster, **and** each ACFS file system, where ACFS Security will be used, must be prepared for ACFS Security as described in Oracle® Automatic Storage Management Administrator's Guide 11g Release 2. It is recommended to use 'asmca' to perform these operations.

These steps can also be done via the command line interface as follows:

To use any of the new ACFS features introduced in 11.2.0.2, ASM and ADVM compatibility attributes must be set to 11.2.0.2.0.

To initialize ACFS security for the cluster, execute as 'root':

```
# /sbin/acfsutil sec init -u secadmin -g secadmingrp
```

where 'secadmin' is an existing OS user designated as the (first) security administrator and 'secadmingrp' is an existing OS group designated as the security administrator group. All security administrators must be part of the security administrator group. This command can only be run by 'root' and it prompts for a password. This is a password for the new security administrator. It is recommended that the 'secadmin' user changes the password immediately after the initialization of ACFS security:

```
$ /sbin/acfsutil sec admin password
```

Once the password of the ACFS security has been changed by 'secadmin', 'root' cannot make any changes to it. Once security is initialized for the cluster, each ACFS file system where security will be used, must be prepared for security:

```
$ /sbin/acfsutil sec prepare -m /opt/oracle/acfsmounts/data_tdevolume
```

where '-m' specifies the mount point where the ACFS file system is mounted.

This example assumes that the TDE Wallet is already created in the directory /opt/oracle/acfsmounts/data_tdevolume/wallet_dir/

To protect the TDE Wallet, first, a realm must be created on the ACFS file system:

```
$ /sbin/acfsutil sec realm create TDEWalletRealm
  -m /opt/oracle/acfsmounts/data_tdevolume
  -d "Realm to protect the TDE Wallet"
  -e off
```

This command creates a realm named 'TDEWalletRealm' on the ACFS file system mounted on '/opt/oracle/acfsmounts/data_tdevolume'. '-e off' option specifies that encryption has been turned off for this realm.

To allow only the Oracle executable access to the TDE Wallet, an application rule can be created as follows:

```
$ /sbin/acfsutil sec rule create allowOracleAppRule
  -m /opt/oracle/acfsmounts/data_tdevolume
  -t application $ORACLE_HOME/bin/oracle
  -o ALLOW
```

'-t' specifies the type of the rule. In this example, it is an application rule. Other types of rules based on the username, time, or hostname are also supported. Once a rule is created, it can be added to a ruleset as follows:

Create a new ruleset:

```
$ /sbin/acfsutil sec ruleset create TDEWalletRuleSet
  -m /opt/oracle/acfsmounts/data_tdevolume
```

Add the rule to the ruleset:

```
$ /sbin/acfsutil sec ruleset edit TDEWalletRuleSet
  -m /opt/oracle/acfsmounts/data_tdevolume
  -a allowOracleAppRule -o ANY_TRUE
```

Create a new rule for Java applications such as 'orapki' and Oracle Wallet Manager ('owm')

```
$ /sbin/acfsutil sec rule create allowJavaAppRule
  -m /opt/oracle/acfsmounts/data_tdevolume
  -t application $ORACLE_HOME/jdk/bin/java –o ALLOW
```

Add the rule to the TDEWalletruleset:

```
$ /sbin/acfsutil sec ruleset edit TDEWalletRuleSet
  -m /opt/oracle/acfsmounts/data_tdevolume
  -a allowJavaAppRule -o ANY_TRUE
```

The TDE Wallet can then be protected by the realm:

```
$ /sbin/acfsutil sec realm add TDEWalletRealm
  -m /opt/oracle/acfsmounts/data_tdevolume
  -u oracle -l ALL:TDEWalletRuleSet –f
  -r /opt/oracle/acfsmounts/data_tdevolume/wallet_dir
```

For more details on all the commands and options supported by the ACFS Security feature, see the Oracle® Automatic Storage Management Administrator's Guide 11g Release 2 (11.2).

Exadata Database Machine

Exadata Database Machine X2, based on Oracle Database 11gR2 (11.2.0.2) is equipped with Intel® Xeon® 5600 CPUs that provide hardware-based cryptographic acceleration. The storage nodes of both X2-2 and X2-8 are identical; they use the Intel® Xeon® L5640 CPU with AES-NI. Decryption takes place on the storage nodes when the Oracle Optimizer decided to push the query down to the intelligent storage cells; if not, data is decrypted on the compute nodes, where only the X2-2 model has the CPUs to accelerate this operation. The Nehalem technology in the Intel® Xeon® X7560 CPU provides 2 x faster cryptographic operations when compared to

Exadata V2, while the Intel® Xeon® 5600 CPUs accelerate encryption up to 6 x faster and decryption up to 8 x faster. The following table illustrates this fact:

EXADATA MODEL	X2-2		X2-8	
NODE	ENCRYPT	DECRYPT	ENCRYPT	DECRYPT
COMPUTE	Enable hardware-acceleration in Intel® Xeon® X5670 with patch 10080579	Hardware acceleration in Intel® Xeon® X5670 enabled by default	Hardware acceleration (~ 2x) through Nehalem technology in Intel® X7560 (encryption acceleration requires patch 10080579)	
STORAGE	n/a	Hardware acceleration in Intel® Xeon® L5640 enabled by default	n/a	Hardware acceleration in Intel® Xeon® L5640 enabled by default

Exadata Database Machine V2 (based on Oracle Database 11g Release 2 (11.2.0.1)) allows encrypting data stored in Exadata while fully supporting the benefits of the 'Smart Scan' functionality, because the Exadata firmware has been extended to reach out to the Database server to decrypt table and tablespace keys when needed:

When TDE **tablespace** encryption is used, data sent to Exadata is (optionally) first compressed with Exadata Hybrid Columnar Compression (EHCC), then encrypted, and finally written to disc.

If TDE **column** encryption is used in Exadata, data is first encrypted, then (optionally) compressed using EHCC (which results in a lower compression ratio of the encrypted column), and then written to disk.

When selecting data, it is first decrypted, then 'smart scan' is applied to remove unneeded data from the result set, then the result set is decompressed, and returned to the database.

Exadata Hybrid Columnar Compression used in conjunction with TDE tablespace encryption will more than compensate the small performance overhead caused by TDE tablespace encryption, because *compressed* data is encrypted: When the size of a compressed data set is 35% smaller than uncompressed, TDE tablespace encryption has to encrypt and decrypt 35% less data.

In Oracle Exadata Database Machine, the 'compute node' is a RAC-enabled database; even though the storage is facilitated using ASM, creating an ACFS file system on top of ASM in Exadata is not supported. Wallet operations that do not update the wallet content (wallet open/close commands) are synchronized between RAC instances, even if the wallet is not stored in a central location, while master key set key or re-key operations are only synchronized with a centralized, shared wallet.

Detailed Upgrade Information

Existing Oracle Database 10g Release 2 TDE implementations can upgrade to Oracle Database 11g easily. The primary consideration to take into account is whether there are plans to deploy an HSM device after upgrade.

TRANSPARENT DATA ENCRYPTION MIGRATION AND UPGRADE MATRIX		
PLANNED DEPLOYMENT AFTER UPGRADE:	UPGRADE TO DATABASE VERSION:	RECOMMENDATIONS
TDE column encryption with software wallet	11.1.0.7	Confirm patch 8421211 is installed.
Migration to HSM for storage of TDE column encryption master key	11.1.0.7	Do not re-key with the original `sqlnet.ora` entry; first change `METHOD=FILE` to `METHOD=HSM`, leaving `DIRECTORY` entry as is, and then migrate column encryption master key to HSM; this automatically creates an additional static TDE tablespace encryption master key inside the HSM. To enable access to encrypted data in undo/redo logs, encrypted export and backup files, convert software wallet to (local) 'auto-open' <u>and</u> move encryption wallet to a secure location outside of the wallet directory, or change the wallet password to the HSM authentication string.
Convert TDE column encryption to TDE tablespace encryption with software wallet after upgrading **existing** ORACLE_HOME	11.1.0.7	Prior to upgrade, decrypt all encrypted application table columns. After upgrade, re-key the TDE master key. This will automatically add a new static TDE tablespace encryption master key to the wallet. Please follow instructions found in 'Moving Your Application Data to Encrypted Tablespaces' on page 8.
Migrate data from an existing Oracle Database 10gR2 database with TDE column encryption to a separate newly created Oracle Database 11g database with TDE tablespace encryption deployed and a new software wallet.	11.1.0.7	Prior to upgrade, decrypt all encrypted application table columns. In the new Oracle Database 11g database, initialize the new wallet. This will transparently create a new TDE column encryption master key and a TDE tablespace encryption master key and store them in the Oracle Wallet. Create new tablespaces using the ENCRYPT syntax. Please note that re-key operations will fail because Oracle Database 11gR1 does not support re-keying tablespace master keys or tablespace keys. Please follow instructions found in 'Tablespace Re-Key Restrictions' on page 9 for instructions on how to re-key an existing encrypted tablespace.

TRANSPARENT DATA ENCRYPTION MIGRATION AND UPGRADE MATRIX		
Migrate data from an existing Oracle Database 10gR2 database that uses TDE column encryption to a separate newly created Oracle Database 11g database with TDE tablespace encryption deployed, storing the TDE tablespace encryption master key in an HSM device and upgrading **existing** ORACLE_HOME	11.1.0.7	Prior to upgrade, decrypt all encrypted application table columns. After the upgrade, do **not** re-key the master key, which would add a TDE tablespace encryption master key to the wallet that cannot be migrated to an HSM. Install and test HSM software according the HSM vendor instructions. Change `METHOD=FILE` to `METHOD=HSM`, leaving `DIRECTORY` entry as is, and then migrate (unused) column encryption master key to HSM, which also creates TDE tablespace encryption master key in HSM. To enable access to encrypted data in undo/redo logs, encrypted export and backup files, convert software wallet to (local) 'auto-open' <u>and</u> move encryption wallet to a secure location outside of the wallet directory, or change the wallet password to the HSM authentication string. Please follow instructions found in 'Moving Your Application Data to Encrypted Tablespaces' on page 8.
TDE Tablespace Encryption with HSM, **new** installation	11.1.0.7	Prior to upgrade, decrypt all encrypted application table columns. Do **not** create software wallet, which would include a TDE tablespace encryption master key that cannot be migrated to HSM. Install and test HSM software according the HSM vendor instructions. Add `ENCRYPTION_WALLET_LOCATION=` `(SOURCE = (METHOD = HSM))` to `sqlnet.ora.` Create TDE tablespace encryption master key in HSM. Please follow instructions found in 'Tablespace Re-Key Restrictions' on page 9 for instructions on how to re-key an existing encrypted tablespace.

ORACLE®

Transparent Data Encryption Best Practices
February 2011
Author: Peter Wahl

Oracle Corporation
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

Worldwide Inquiries:
Phone: +1.650.506.7000
Fax: +1.650.506.7200
oracle.com

Oracle is committed to developing practices and products that help protect the environment

0109