

An Oracle White Paper
August 2009

Oracle Advanced Security Transparent Data Encryption Best Practices

Introduction	1
Important Concepts.....	1
Transparent Data Encryption Key Architecture	3
Key Generation and Backup	3
Key Storage	3
Key Exchange / Rotation.....	4
Oracle Wallet Directory and File Permissions	4
(Local) Auto-Open vs. Encryption Wallet	5
Strong Wallet Password.....	6
Split knowledge about the Wallet password.....	6
Changing the Wallet Password	7
Using Hardware Security Modules (HSM)	7
TDE Tablespace Encryption	9
Oracle Applications certified for TDE Tablespace Encryption.....	9
Moving Your Application Data to Encrypted Tablespaces	10
Tablespace Re-Key Restrictions.....	10
TDE Column Encryption.....	11
Oracle Application certified for TDE Column Encryption.....	11
Identifying Sensitive Columns	12
Encrypting indexed columns	12
Reducing the storage overhead	12
Encrypting Columns in Gigabyte and Terabyte Tables.....	13

TDE Tablespace Encryption or TDE Column Encryption?	14
Clear-Text Copies of Encrypted Data	14
Attestation	15
Oracle Data Guard	15
Physical Standby.....	15
Logical Standby.....	16
Oracle Streams	16
Oracle Transparent Data Encryption and Oracle RMAN	17
Real Application Clusters (RAC)	17
Detailed Upgrade Information	18

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 helps address compliance 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 such as credit card and social security numbers. Oracle Advanced Security TDE tablespace encryption and support for hardware security modules (HSM) were introduced with Oracle Database 11g.

Important Concepts

- **Master Key** – The encryption key used to encrypt secondary keys used for column encryption and tablespace encryption. Master keys are part of the Oracle Advanced Security two-tier key architecture.
- **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 key.
- **Tablespace Key** – The key used to encrypt a tablespace. These keys are encrypted using the tablespace master key and are stored in the tablespace header of the encrypted tablespace, as well as, for performance reasons, in the header of each database 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 using an administratively defined password.
- **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.

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): An 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 and whenever the master key is changed. The wallet is a critical component and should be backed up in a secure location, on-site and offsite. Note that if you are using Oracle Data Guard you should copy the wallet to the secondary site for smooth transition in the case of failover. If you are using an HSM device, follow the manufacturer's instructions for insuring the recoverability of keys in case the HSM fails. This may involve the secure export of keys out of the HSM and/or authentication of multiple administrators for key recovery.

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	Column Encryption	Yes	No
Oracle Database 11gR1 (11.1.0.6)	Column Encryption	Yes	Yes
	Tablespace Encryption	Yes	No
Oracle Database 11gR1 (11.1.0.7)	Column Encryption	Yes	Yes
	Tablespace Encryption	Yes	Yes (no re-key)

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 11g, 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.

Note the restriction 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.

RE-KEY SUPPORT				
	TDE COLUMN ENCRYPTION		TDE TABLESPACE ENCRYPTION	
	MASTER KEY	TABLE KEYS	MASTER KEY	TABLESPACE KEYS
Re-key support	Yes	Yes	No	No

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 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. Oracle recommends using the Oracle Wallet Manager default directory:

```
/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' 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 operations against the wallet.

(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. 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 or remove the original encryption wallet. Re-keying the master key requires the original encrypted wallet to be present. When the master key is re-keyed the corresponding auto-open wallet is automatically updated

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 Oracle Database 11gR1 with Enterprise Manager Database Control 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, 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:

- 1.) Create a user ‘Sentinel’ in your database with only ‘create session’ and ‘alter system’ privileges
- 2.) Create a Secure External Password Store (SEPS), following the instructions in this document:
http://www.oracle.com/technology/deploy/security/database-security/pdf/twp_db_security_secure_ext_pwd_store.pdf
 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`’
- 3.) 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
```

- 4.) 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 (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 requires 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 using 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 TDE tablespace encryption (from 11.1.0.7) 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 column or tablespace keys to the HSM device, where they are decrypted and returned to the database to process encrypted data.

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.

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, you can migrate the existing master key from the wallet 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)))
```

To migrate the TDE column encryption master key to HSM:

```
SQL> alter system set encryption key identified by
"HSM_username:HSM_password" 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. After this command completes successfully, the software wallet can be backed up and `sqlnet.ora` can be simplified to:

```
ENCRYPTION_WALLET_LOCATION =
  (SOURCE = (METHOD = HSM))
```

However, if you still have data protected by a wallet-based master key, you need to retain the original software wallet. This will be the case if you performed a re-key after upgrading to Oracle Database 11g, but before changing the `sqlnet.ora` entry to `METHOD = HSM`, if 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. The software wallet can be changed to

an auto-login wallet, or the wallet password must be changed to match the full HSM credential "HSM_username:HSM_password".

New in Oracle Database 11.1.0.7 is the ability to create and store the master key for TDE Tablespace Encryption 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 11g Release 1. 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.

Please note that on some 64-bit operating systems, attempts to load the PKCS#11 library failed; this behavior can be corrected by applying patch 8211698 from <http://updates.oracle.com/download/8211698.html>

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

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:

- Oracle E-Business Suite 11.5.10 (see Oracle Metalink Note 828223.1)
- Oracle E-Business Suite 12.0.4 (see Oracle Metalink Note 828229.1)

- Oracle PeopleSoft Enterprise 8.48 and later
- Oracle Siebel CRM
- Oracle JD Edwards EnterpriseOne

Internal benchmark tests and pilot 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 an existing un-encrypted tablespace to a new encrypted tablespace using these steps:

- Export all application tablespaces with Oracle Data Pump Export ('expdp'), optionally compressing the dump file for faster processing and less storage for the dump file
- Backup the database using your standard backup procedures
- Convert clear text tablespaces to encrypted tablespaces
 - Using `'dbms_metadata.get_ddl'`, extract the original commands 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, `'including contents and datafiles'`
 - Create the encrypted tablespaces by running the edited script
- Import all application tablespaces with Oracle Data Pump Import ('impdp')
- Verify application is working properly

Tablespace Re-Key Restrictions

Oracle Database 11g doesn't 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, Oracle recommends using the Oracle Data Pump utility to extract the application data from the encrypted tablespaces, create new encrypted tablespaces, and then re-importing the data into the new encrypted tablespace using Oracle Data Pump.

- 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).
- Backup the database using your standard backup procedures.

- 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 11gR1 (10.2.0.4 and 11.1.0.7 are recommended):

- Oracle E-Business Suite 11.5.9 (see Oracle Metalink Note 403294.1)
- Oracle PeopleSoft Enterprise 8.46 and later
- Oracle Siebel CRM
- SAP (customers and partners can refer to SAP note 974876)

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 dba_tab_cols where
column_name like '%SSN%' or
column_name like '%SECNUM%' or
column_name like '%SOCIAL%' and owner='OWNER';
```

The value for 'owner' would be 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. By default, TDE column encryption adds 'salt' and an integrity check (Message Authentication Code) 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 dangerous when uniqueness of the values in a column is not given. For example, when a column contains sensitive patient information about a rare disease: most patients would enter 'No', while only a few would enter 'Yes'. All of the cipher texts for 'Yes' would match. To get beyond this limitation, we encrypt the plaintext with 'salt', a random,

16-byte string used to modify each clear text value. 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, 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 Database 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 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, and so on. This cannot be changed and is independent of the encryption algorithm used: 3DES168, AES128, AES192 (default), or AES256.

It is highly recommended to install patch 7639262 from

<http://updates.oracle.com/download/7639262.html> for TDE column encryption in 10.2.0.4 and <http://updates.oracle.com/download/8421211.html> 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.

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 ...:
Keys need to be rotated on a semi frequent basis	Key rotation is not required
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.

Oracle Transparent Data Encryption and Oracle RMAN

ORACLE RMAN ENCRYPTION AND COMPRESSION OF DATA ENCRYPTED BY TDE

	BACKUP WITH COMPRESSION	BACKUP WITH ENCRYPTION	BACKUP WITH COMPRESSION AND ENCRYPTION
Data not encrypted	Data compressed	Data encrypted	Data compressed first, then encrypted
Data encrypted with TDE column encryption	Data compressed; encrypted columns are treated as if they were not encrypted.	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 blocks are decrypted, compressed, and re-encrypted; Clear text blocks 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 securely share the master encryption key.

Real Application Clusters (RAC)

When Oracle Advanced Security TDE (both column-level and tablespace encryption) 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. 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 re-keyed on the first 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 be corrupted when one instance re-keys the master key without the other instances being properly updated. 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.

Re-keying the master key in an HSM device is easier compared to local software wallets, since the master key is stored in a central location, and the master key is no longer stored in memory of the individual instances; simply re-key the master key on the first instance; this decrypts the table keys (shared by all RAC instances) with the old master key and re-encrypts them with the new master key; all other instances will simply pass the newly encrypted table keys to the HSM device, where they are decrypted with the new master key.

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 external security module such as 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	11.1.0.6 or 11.1.0.7	If the TDE wallet has been created in Oracle Database 10g Release 2 and you wish to use HSM mode at some point, it is not recommended to perform a master key re-key operation after upgrade to Oracle Database 11g, until the entry in 'sqlnet.ora' has been changed to 'METHOD = HSM'. Prior to changing this setting, performing a master key re-key operation after upgrade will add an additional master key for TDE tablespace encryption to your wallet, which cannot be migrated to HSM.
Migration to HSM for storage of TDE column encryption master key	11.1.0.6 or 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. Convert software wallet to 'auto-open' or change password to HSM authentication string to enable access to encrypted data in redo logs and from backup tapes.
Convert TDE column encryption to use TDE tablespace encryption with software wallet after upgrading existing ORACLE_HOME	11.1.0.6 or 11.1.0.7	Prior to upgrade, decrypt all encrypted application table columns. After upgrade, re-key the TDE table master key. This will transparently add a new TDE tablespace encryption master key to wallet. Please follow instructions found in 'Moving Your Application Data to Encrypted Tablespaces' on page 8.

TRANSPARENT DATA ENCRYPTION MIGRATION AND UPGRADE MATRIX		
Migrate data from an existing Oracle Database 10g Release 2 database that uses TDE column encryption to a separate newly created Oracle Database 11g database with TDE tablespace encryption deployed and a new software wallet.	11.1.0.6 or 11.1.0.7	<p>Prior to upgrade, decrypt all encrypted application table columns.</p> <p>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 11g 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.</p>
Migrate data from an existing Oracle Database 10g Release 2 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	<p>Prior to upgrade, decrypt all encrypted application table columns.</p> <p>After the upgrade, do not re-key the master key, which would add a TDE tablespace encryption master key to the wallet, which 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. Convert software wallet to 'auto-open' or change wallet password to HSM authentication string to enable access to encrypted data in redo logs and from backup tapes. Please follow instructions found in 'Moving Your Application Data to Encrypted Tablespaces' on page 8.</p>
TDE Tablespace Encryption with HSM, new installation	11.1.0.7	<p>Prior to upgrade, decrypt all encrypted application table columns.</p> <p>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=</p> <pre>(SOURCE = (METHOD = HSM)) to sqlnet.ora.</pre> <p>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.</p>



Transparent Data Encryption Best Practices
August 2009 (Version 11)
Author: Peter A. 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

Copyright © 2009, Oracle and/or its affiliates. All rights reserved. This document is provided for information purposes only and the contents hereof are subject to change without notice. This document is not warranted to be error-free, nor subject to any other warranties or conditions, whether expressed orally or implied in law, including implied warranties and conditions of merchantability or fitness for a particular purpose. We specifically disclaim any liability with respect to this document and no contractual obligations are formed either directly or indirectly by this document. This document may not be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without our prior written permission.

Oracle is a registered trademark of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.