10g R2 New Feature TDE : Transparent Data Encryption [ID 317311.1]

Modified 06-JUL-2011   *Type* BULLETIN   *Status* PUBLISHED

In this Document

Applies to:

Oracle Server - Enterprise Edition - Version: 10.2.0.1 to 11.2.0.2 - Release: 10.2 to 11.2
Information in this document applies to any platform.
Checked for relevance on 12-Jan-2011

Purpose

This document explains how to use the Transparent Database Encryption (TDE) new feature in 10g R2 that provides an additional layer of security by transparently encrypting column data stored on disk.

Scope and Application

For all DBAs and end-users willing to hide confidential information by encrypting data.

10g R2 New Feature TDE : Transparent Data Encryption

Licensing

Transparent Data Encryption (TDE) belongs to the Advanced Security Option that is available as an Option for the Oracle Database Enterprise Edition only. See also the Oracle Database 11g: Product Editions & Features page where Transparent Data Encryption is listed with Oracle Advanced Security. Note that in this reference *Data Encryption Toolkit* refers to the selective data encryption features available with DBMS_CRYPTO and DBMS_OBFUSCATION_TOOLKIT that can be used without additional licensing in all product editions.

Concepts

Transparent data encryption is a key-based access control system. Even if the encrypted data is retrieved, it cannot be understood until authorized decryption occurs, which is automatic for users authorized to access the table. This means that TDE is no replacement for normal access control, any database user that has the proper select privileges on a certain table can still query the rows even if they are encrypted using TDE. TDE protects your data from unauthorized access that may occur outside the scope of the designated instance such as direct disk access to the database host server and backup media that contains copies of your datafiles.

Key Management

The keys for all tables containing encrypted columns are encrypted with the database server master key and stored in a dictionary table in the database. The TDE master encryption key is an AES256 key. No keys are stored in the clear. The master key is stored in the wallet file. Table keys are encrypted by default with AES192 but this can be changed. When you create a wallet with the statement mentioned in step A.4 below, the master key is generated automatically using a secure random number generator, it has no relationship with the wallet password that you specify with the IDENTIFIED BY clause. Particularly this means that if you create a new wallet even with the same wallet password, you will generate a new master key that cannot be used to decrypt data that was encrypted using a previous wallet.

WARNING

If you lose the wallet or wallet password and you are not able to open the wallet, then Oracle Support Services cannot help you to recover the encrypted data. There are no back doors to access TDE encrypted data from the database without the original wallet that holds the master key. Also Oracle cannot help you recover a lost wallet password.

Setup

What is needed to create/store the master key of the server ?

A wallet file that is outside the database and accessible only to the security administrator:

1. Storing the master key in this way prevents its unauthorized use.
2. It is also used to generate encryption keys and perform encryption and decryption.

A. Prepare the database to allow Transparent Data Encryption

1- Specify the location of the wallet file used to store the encryption master key by adding the following entry in $ORACLE_HOME/network/admin/sqlnet.ora:

```
     ENCRYPTION_WALLET_LOCATION=
          (SOURCE=(METHOD=FILE)(METHOD_DATA=
                  (DIRECTORY=D:\Oracle\product\10.2.0\db_2)))
```

You can enter any existing directory, but make sure there is no obfuscated wallet in this directory (named 'cwallet.sso')

> If not specified, when "ALTER SYSTEM SET [ENCRYPTION] KEY ..." is executed, the server may create the wallet in the default location ($ORACLE_BASE/ADMIN/<SID>/WALLET on certain platforms , another directory on other platforms; refer Bug:4956266).
>
> On certain platforms, this entry is mandatory : otherwise the creation fails with ORA-28368 .
> If this location doesn't exist, an ORA-28368 is also received.
>
> This default location can help a lot when someone needs to have different wallets for different instances in the same ORACLE_HOME.
>
> Instead of playing around with different values of TNS_ADMIN environment variable, let each instance write its own wallet in its default location.

2- Start the listener:

```
lsnrctl start
```

3- Connect to the database 'as sysdba':

```
sqlplus '/as sysdba'
```

4. Create the encrypted wallet file (the ENCRYPTION clause is optional):

```
SQL> ALTER SYSTEM SET ENCRYPTION KEY IDENTIFIED BY "welcome1";
```

If no encrypted wallet is present in the directory defined in sqlnet.ora :

1.) It creates an encrypted wallet (ewallet.p12)
2.) It opens the wallet
3.) It creates the database server master encryption key for TDE

If an encrypted wallet already exists :

1.) It opens the wallet
2.) It creates or re-creates the database server master encryption key for TDE.

This is usually done ONLY ONCE unless you want to re-encrypt your data with a new encryption key for the whole database. Also do NOT repeat this statement on different RAC nodes, instead, copy the wallet over to the other nodes or make it available by pointing sqlnet.ora parameter ENCRYPTION_WALLET_LOCATION to a shared drive. The master key is automatically created with a secure random key generator, the password you supply ('welcome1' in the above example) is just the wallet password. The wallet password can be changed later using Oracle Wallet Manager (owm) without affecting the master key, do this only when the wallet is closed.

If you fail to enclose the wallet password in double quotes like in the above example, the actual wallet password will be in uppercase, this is because the wallet password follows the same rule as any identifier such as an object name when being parsed by the sql engine. The same is true when you open the wallet.

> Note: It can also be an existing key pair from a PKI certificate designated for encryption. To use transparent data encryption with PKI key pairs, the issuing certificate authority must be able to issue X.509v3 certificates with the key usage field marked for encryption.

```
SQL> ALTER SYSTEM SET ENCRYPTION KEY "certificate_ID"  IDENTIFIED BY "welcome1";
```

You can search for a certificate_ID by querying the V$WALLET fixed view when the wallet is open. Only certificates that can be used as master keys by transparent data encryption are shown.

Remark: for a certificate to work this way, the certificate authority must not use any Key Usage specification to sign the certificate. If you have problems seeing the certificate in V$WALLET, create a new certificate without any Key Usage specifications.
To verify the certificate, export it from the wallet using OWM and give the file an extension .crt.

For example: mycertificate.crt,
then double-click the certificate, or right-click the certificate,
then click Open , the Certificate dialog box appears,
go to [Details] and Show: [Extensions Only],
this must only include: [Basic Constraints, Netscape Comment, Subject Key Identifier, Authority Key Identifier]
and not [Key Usage] .

Note from the openssl.cnf file in the header for [usr_cert] , we have key Usage left commented out.

For later sessions, you do not have to use the same command : you only need to open the wallet (it is automatically closed when you shut down the database).
To load the master key after the database is restarted :

```
SQL> ALTER SYSTEM SET WALLET OPEN IDENTIFIED BY  "welcome1";
```

This statement must be repeated for each instance of a RAC database, if the wallet file ewallet.p12 is on a shared location it doesn't need to be copied, however it is recommended to store a copy of the wallet file on each node and restrict maintenance operations on the wallet to an exclusively mounted database and then to copy the wallet over, it would be a good practice at this time to also make a backup of the wallet and to keep this wallet backup separate from the datafile backups.

The master encryption key is necessary because each table has its own encryption key. The column keys are stored in the database and encrypted with the master key.

When the wallet is created for the first time with the *ALTER SYSTEM SET ENCRYPTION KEY IDENTIFIED BY "<wallet_password>"*; command it generates a master key using the Advanced Encryption Standard with 256 bits. Note that only users with the 'ALTER SYSTEM' privilege ('/as sysdba' or users with the default DBA role, like 'system') can create a wallet and a master key or open the wallet. Users without proper privileges are not able to use the key, since it is not available to them.

B. The database is ready to allow encryption: how to encrypt/decrypt data in table columns.

1- Create table SCOTT.CUST_PAYMENT_INFO with encryption (default AES192) applied to the CREDIT_CARD_NUMBER column .

NO SALT is specified since there will be an index on the credit_card_number column, which is not possible when the encrypted values are 'salted'.

Salt is a way to strengthen the security of encrypted data. It is a random string added to the data before it is encrypted, causing repetition of text in the clear to appear different when encrypted. Salt thus removes one method attackers use to steal data, namely, matching patterns of encrypted text.

```
SQL> conn system/manager
```

```
SQL> create user SMITH identified by welcome1;
```

```
SQL> grant create session to SMITH;
```

```
SQL> conn scott/tiger
```

```
SQL> create table CUST_PAYMENT_INFO
                    (first_name varchar2(11),
                     last_name varchar2(10),
                     order_number number(13),
                     CREDIT_CARD_NUMBER varchar2(20) ENCRYPT NO SALT);
```

2- Insert data into SCOTT.CUST_PAYMENT_INFO

```
SQL> insert into cust_payment_info values ('Jon', 'Oldfield', 10001, '5446-9597-0881-2985');
```

```
SQL> insert into cust_payment_info values ('Chris', 'White', 10002, '5122-3580-4608-2560');
```

```
SQL> insert into cust_payment_info values ('Alan', 'Squire', 10003, '5595-9689-4375-7920');
```

3- Grant select privilege to end user :

```
 SQL> grant select on SCOTT.CUST_PAYMENT_INFO to SMITH;
```

4- Connect as the end user and find out if the decryption works:
SQL> select * from SCOTT.CUST_PAYMENT_INFO;

```
SQL> conn SMITH/welcome1
```

Three things happen here that show the true transparency of TDE:

1.) An index has been created on the encrypted column.
2.) Even if the credit card numbers are stored encrypted, the select displays numbers in clear text.

The owner of the table SCOTT confirms that all settings are correct by:

```
SQL> conn SCOTT/tiger
```

```
SQL> select * from USER_ENCRYPTED_COLUMNS;
```

```
TABLE_NAME         COLUMN_NAME         ENCRYPTION_ALG    SAL
------------------ ------------------- ----------------  ---
CUST_PAYMENT_INFO  CREDIT_CARD_NUMBER  AES 192 bits key  NO
```

5- In which situations an encrypted column cannot be decrypted ?

5.1- We have 2 wallet files :
--> an old one ewallet.p12.old (key=welcome1)
--> a new one ewallet.p12   (key=test1)

```
SQL> alter system set wallet open identified by "test1";
System altered.
SQL> conn scott/tiger
Connected.
```

```
SQL> select * from welcome;
```

```
select * from welcome
              *
      ERROR at line 1:
      ORA-28362: master key not found
```

The table WELCOME had been created with column C encrypted with 'welcome1' master key.

```
SQL> connect / as sysdba
Connected.
SQL> alter system set wallet close;
System altered.
```

Rename the new wallet file name (key=test1) to ewallet.p12.old and set the old wallet file as the current one :

```
SQL> alter system set wallet open identified by "welcome1";
System altered.
```

```
SQL> conn scott/tiger
Connected.
```

```
     SQL> select * from scott.welcome;
```

```
         C        SN
  ---------- ----------
         1
```

5.2- The wallet has been opened and closed during the instance life:

```
SQL> select * from scott.welcome;

         C        SN
  ---------- ----------
         1

 SQL> alter system set wallet close;
  System altered.

  SQL> select * from scott.welcome;
  select * from scott.welcome
                     *
  ERROR at line 1:
  ORA-28365: wallet is not open
```

5.3- The master key has been reset :

```
SQL> conn / as sysdba
Connected.
SQL> ALTER SYSTEM SET ENCRYPTION KEY IDENTIFIED BY "tde1";
System altered.

SQL> create table SCOTT.TDE (c number ENCRYPT);
Table created.

SQL> insert into  SCOTT.TDE values (1);
1 row created.

SQL> insert into  SCOTT.TDE values (2);
1 row created.
```

```
        SQL> commit;
        Commit complete.

        SQL> select * from SCOTT.TDE;

          C

 ----------

          1

          2

        SQL> select * from DBA_ENCRYPTED_COLUMNS;

OWNER  TABLE_NAME        COLUMN_NAME        ENCRYPTION_ALG   SAL
------ ----------------- ------------------ ---------------- ---
 SCOTT  CUST_PAYMENT_INFO CREDIT_CARD_NUMBER AES 192 bits key NO
 SCOTT  TDE               C                  AES 192 bits key YES
 SCOTT  WELCOME           C                  AES 128 bits key YES
```

Rename ewallet.p12 file to ewallet.p12.v1

```
SQL> ALTER SYSTEM SET ENCRYPTION KEY IDENTIFIED BY "tde2";
System altered.

SQL> insert into  SCOTT.TDE values (3);
insert into  SCOTT.TDE values (3)
                   *
ERROR at line 1:
ORA-28362: master key not found

SQL> ALTER TABLE scott.tde MODIFY (c DECRYPT);
ALTER TABLE scott.tde MODIFY (c DECRYPT)
*
ERROR at line 1:
ORA-28362: master key not found
```

Now it is only possible to decrypt the data if you still can open the ewallet.p12.v1

What SHOULD HAVE BEEN done before renaming the ewallet.p12 file ?

5.3.1 SQL> ALTER TABLE scott.tde MODIFY (c DECRYPT);
       or export with expdp without ENCRYPTION_PASSWORD (refer note:317317.1)

5.3.2 Rename ewallet.p12 file to ewallet.p12.v1

5.3.3 SQL> ALTER SYSTEM SET ENCRYPTION KEY IDENTIFIED BY "tde2";

5.3.4 SQL> ALTER TABLE scott.tde MODIFY (c ENCRYPT);
       or import with impdp without ENCRYPTION_PASSWORD (refer note:xxx)

5.3.5 SQL> insert into SCOTT.TDE values (3);
       SQL> select * from SCOTT.TDE;

Regenerate the master key only if it has been compromised.
Frequent master key regeneration does not necessarily enhance system security. Security modules can store a large, but not infinite, number of keys, and frequent master key regeneration can exhaust all the available storage space.

6- Which operations can be performed on encryption attribute of a column ?

6.1- Change the encryption algorithm for the columns in a table to use less storage space :

```
SQL> alter table scott.tde REKEY USING '3DES168';
Table altered.

SQL> alter table scott.tde REKEY USING 'AES128';
Table altered.
```

You see that only one algorithm can be used per table as no column name can be specified.

6.2- Remove the encryption attribute of a column :

```
SQL> alter table scott.tde MODIFY (c DECRYPT);
```

6.3- Change the SALT attribute on an encrypted column when you want to add an index on this column:

```
SQL> alter table scott.tde MODIFY (c ENCRYPT NO SALT);
Table altered.
SQL> create index scott.i_tde on scott.tde (c);
Index created.

SQL> alter table scott.tde MODIFY (c ENCRYPT SALT);
alter table scott.tde MODIFY (c ENCRYPT SALT)
                      *
ERROR at line 1:
ORA-28338: cannot encrypt indexed column(s) with salt
```

C. How does LogMiner display the encrypted data

Since Transparent Data Encryption is done right before the data is written (in the SQL layer, to be more precisely) and

transparent to all applications using this layer, there is no easy way to verify that data has truly been encrypted, but Oracle

LogMiner records what is written to disk with the corresponding SQL statements.

1- First, locate your log files. You can find them with:

```
SQL> select member as LOG_FILE_LOCATION from v$logfile;
```

In our example, the output of this query is:

```
SQL> LOG_FILE_LOCATION

------------------------------------

/orcl/redo03.log

/orcl/redo02.log

/orcl/redo01.log
```

Each of them needs to be registered into LogMiner for later processing :

```
SQL> EXECUTE DBMS_LOGMNR.ADD_LOGFILE('/orcl/redo01.log',DBMS_LOGMNR.NEW);
SQL> EXECUTE DBMS_LOGMNR.ADD_LOGFILE('/orcl/redo02.log', DBMS_LOGMNR.ADDFILE);
SQL> EXECUTE DBMS_LOGMNR.ADD_LOGFILE('/orcl/redo03.log', DBMS_LOGMNR.ADDFILE);
```

2- The next step is to start LogMiner and query the log files:

```
    SQL> EXECUTE DBMS_LOGMNR.START_LOGMNR (-
            options => DBMS_LOGMNR.DICT_FROM_ONLINE_CATALOG -
                       + DBMS_LOGMNR.COMMITTED_DATA_ONLY);

    SQL> select sql_redo from v$logmnr_contents
              where table_name = 'CUST_PAYMENT_INFO'
              and operation='INSERT';

  SQL_REDO

  ------------------------------------------------------------

  insert into

  "SCOTT"."CUST_PAYMENT_INFO"("FIRST_NAME","LAST_NAME",

  "ORDER_NUMBER","CREDIT_CARD_NUMBER") values

  ('Jon','Oldfield','10001',Unsupported Type);
```

LogMiner does not support encrypted data, so the encrypted values in the CREDIT_CARD_NUMBER column are

displayed as "Unsupported Type".

If the output doesn't show what you see here, please connect '/as sysdba' and issue the following command only ONCE:

```
    SQL> alter database add supplemental log data;
```

3- To show the difference, drop the table, re-create it with no columns encrypted and check the log files again:

```
    SQL> conn scott/tiger
    SQL> drop table cust_payment_info;
    SQL> create table cust_payment_info
              (first_name varchar2(11),
               last_name varchar2(10),
               order_number number(13),
               credit_card_number varchar2(20));

    SQL> insert into cust_payment_info values
          ('Jon', 'Oldfield', 10001, '5446-9597-0881-2985');
    SQL> insert into cust_payment_info values
          ('Chris', 'White', 10002, '5122-3580-4608-2560');
    SQL> insert into cust_payment_info values
          ('Alan', 'Squire', 10003, '5595-9689-4375-7920');

    SQL> conn / as sysdba

    SQL> EXECUTE DBMS_LOGMNR.ADD_LOGFILE('/orcl/redo01.log', DBMS_LOGMNR.NEW);
    SQL> EXECUTE DBMS_LOGMNR.ADD_LOGFILE('/orcl/redo02.log', DBMS_LOGMNR.ADDFILE);
    SQL> EXECUTE DBMS_LOGMNR.ADD_LOGFILE('/orcl/redo03.log', DBMS_LOGMNR.ADDFILE);

    SQL> EXECUTE DBMS_LOGMNR.START_LOGMNR(-
               options => DBMS_LOGMNR.DICT_FROM_ONLINE_CATALOG -
                          + DBMS_LOGMNR.COMMITTED_DATA_ONLY);

    SQL> select sql_redo from v$logmnr_contents
              where table_name = 'CUST_PAYMENT_INFO'
             and operation='INSERT';

  SQL_REDO
```

```
-----------------------------------------------------------

insert into

"SCOTT"."CUST_PAYMENT_INFO"("FIRST_NAME","LAST_NAME",

"ORDER_NUMBER","CREDIT_CARD_NUMBER") values

('Jon','Oldfield','10001','5446-9597-0881-2985');
```

The column is not encrypted, and LogMiner shows the clear text data which has been written to disk.

D. Cloning / moving an encrypted database to a new machine :

1- Copy the wallet in the ENCRYPTION_WALLET_LOCATION of the target
2- Open the database
3- Open the wallet :

```
SQL> ALTER SYSTEM SET WALLET OPEN IDENTIFIED BY "welcome1";
```

E. Support for Dataguard :

Data Guard can support TDE on logical standby starting with 11gR1. Please refer to the documentation:
Oracle� Data Guard Concepts and Administration
11g Release 1 (11.1)
Part Number B28294-03
Appendix C 'Data Type and DDL Support on a Logical Standby Database' section 'Support for Transparent Data Encryption (TDE)'

F. Restrictions

--> Distribution/Replication

When asynchronous SQL-based replication is used, information describing the update to be performed is encoded (but not encrypted) in a LOB and can remain in the deferred queue for some time.
While in the deferred queue, the data is not protected.
To minimize this risk, configure replication so that encrypted table information does not remain in the deferred queue for a long time.

--> RAC : database level : key shared amongst instances

--> No exp/imp: only DataPump expdp/impdp

--> TTS :transportable tablespaces

```
$ expdp system/manager transport_tablespaces=users
        directory=d_dir dumpfile=filefull.dmp
        encryption_password="x"

Export: Release 10.2.0.0.0 - Beta on Tuesday, 14 June, 2005 16:02:08

Copyright (c) 2003, Oracle.  All rights reserved.

Connected to: Oracle Database 10g Enterprise Edition Release 10.2.0.0.0 - Beta
With the Partitioning, Oracle Label Security, OLAP and Data Mining options
ORA-39005: inconsistent arguments
ORA-39032: function ENCRYPTION_PASSWORD is not supported in TRANSPORTABLE jobs
```

--> Logminer (see example above in C.)

--> External tables : only with oracle_datapump driver

```
SQL> CREATE TABLE scott.tde_extract (c encrypt, c2)
      ORGANIZATION EXTERNAL
      (TYPE ORACLE_DATAPUMP
            DEFAULT DIRECTORY d_dir
```

```
                    LOCATION ('tde.extract')
              )
              reject limit unlimited
              AS
                select * from scott.tde;

        Table created.


        SQL> select * from scott.tde_extract;

           C         C2

   ---------- ----------

           1        100

    ...



        SQL> select * from DBA_ENCRYPTED_COLUMNS;

  OWNER  TABLE_NAME  COLUMN_NAME ENCRYPTION_ALG    SAL

  ------ ----------- ----------- ----------------  ---

  SCOTT  TDE_EXTRACT C           AES 192 bits key  YES
```

--> Partitioned tables

```
        SQL> create table SCOTT.TDE_PART
                (product_id  number(5) ENCRYPT,
                 time_id     date)
             partition by range (product_id)
             (partition p1 values less than (500),
              partition p2 values less than (maxvalue))  ;

        partition by range (product_id)
                           *
        ERROR at line 4:
        ORA-28346: an encrypted column cannot serve as a partitioning column
```

--> no lob (BLOB and CLOB)

--> no FK nor PK being referenced

```
        SQL> create table scott.tde_fk (c number references scott.tde(c));
        create table scott.tde_fk (c number references scott.tde(c))
                                                             *
        ERROR at line 1:
        ORA-28335: referenced or referencing FK constraint column
                   cannot be encrypted
```

--> no Index types other than B-tree

```
        SQL> create bitmap index scott.i_tde on scott.tde(c);
        create bitmap index scott.i_tde on scott.tde(c)
                                                     *
        ERROR at line 1:
        ORA-28337: the specified index may not be defined on an
                   encrypted column
```

--> Range scan search through an index

```
    --> AQ (Advanced Queueing)
    --> Object Types
    --> Opaque Types
    --> Binary Double  (fixed post 10g R2 in patch set)
    --> Binary Float   (fixed post 10g R2 in patch set)

   Use the DBMS_CRYPTO package instead for your encryption needs in the cases  listed above.
```

References

BUG:4956266 - ORA-28353 WHEN CREATING WALLET
NOTE:317317.1 - 10gR2: How to Export/Import with Data Encrypted with Transparent Data Encryption (TDE)
NOTE:395252.1 - Fails To Open / Create The Wallet: ORA-28353
NOTE:403294.1 - Using TDE Column Encryption with Oracle E-Business Suite Release 11i
http://www.oracle.com/technetwork/database/security/tde-faq-093689.html
http://www.oracle.com/technetwork/database/security/twp-transparent-data-encryption-bes-130696.pdf

Related

Products

- Oracle Database Products > Oracle Database > Oracle Database > Oracle Server - Enterprise Edition

Keywords

CERTIFICATE; CHANGE PASSWORD; ENCRYPTION; ENCRYPTION KEY; PASSWORD; SQLNET.ORA; TRANSPARENT DATA ENCRYPTION

Errors

ORA-39032; ORA-28335; ORA-28362; ORA-28337; ORA-28365; ORA-28368; ORA-28338; ORA-39005; ORA-28346

Back to top