The Encryption Wizard for Oracle

API Library Reference

For Oracle 10g through 19c Databases

Version 9

The Encryption Wizard for Oracle

API Library Reference

Relational Database Consultants, Inc. (RDC)
12021 Wilshire Blvd
Suite 108
Los Angeles, CA. 90025
310-281-1915

www.relationalwizards.com

API Library Reference

Table of Contents

Table of Contents (continued)

Section One

Object Management

The Encryption Wizard for Oracle provides a rich library of Object Management methods that can be invoked from SQL*Plus or directly through any interface such as JDBC that connects to Oracle and can call Oracle stored procedures and packages.

The Object Management API is only to be used for Encryption Wizard Administrators. End-user applications that need to access encrypted data should interface through the Runtime Methods as discussed in the next section. You must be connected to the Oracle RDBMS as *rdc_encrypt_user* to use the object management methods in this section.

The Object Management API allows you to Encrypt and Decrypt data within your Oracle RDBMS. Along with basic encryption functions, the Object Management API provides easy methods to create Runtime Passwords, Decrypted Views, Restricted User Lists, and define audit trails against given sets or subsets or your encrypted data.

It is important to read the Encryption Wizard User Manual before attempting to use any of these API calls. In the User Manual, the concepts of the Encryption Wizard and its underlying API are discussed in more general terms.

If you plan to invoke the Encryption Wizard API from SQL*Plus, it is recommended that you issue this command to start each session:

```
SQL>Set ServerOutput On
```

In this way you can view the informational messages of the Encryption Wizard packages if there is an error or warning. All Encryption Wizard errors and warnings are populated in the table *Internal_Audit* and can be queried using SQL. If you plan on using the Encryption Wizard API on a regular basis, it is helpful to create a series of .sql scripts to run the library commands you plan to use.

1. EncryptData

EncryptData is the core method that is used to encrypt your database data. EncryptData will encrypt every row of a given schema, table or column based on the *key value* (password) that you pass into the method. If you do not provide a key-value, one will be generated for you. If you are using an *administrative password,* EncrpytData will not work without first calling the *login* method.

Along with *scope of encryption*, defined at the schema, table or column level, the method EncrpytData will encrypt table data is based on an *encryption* type and an optional flag to add salt. Valid encryption types can be found querying the table *valid_encryption_types*. EncryptData allows you to specify a c*ommit point*. The c*ommit point* specifies the number of rows to be encrypted before each *commit* is issued to the database.

The method *CountEncryptedColumns* should be called before and after a call to this method to monitor the current state of encryption. EncryptData only returns a message if there is a fatal error.

Specification:

```
Procedure EncryptData
(
        KeyValue            In Varchar2,
        TableOwner          In Varchar2,
        TableName           In Varchar2         Default Null,
        ColumnName          In Varchar2         Default Null,
        EncryptionType      In Varchar2         Default DefaultEncryptionType,
        SaltFlag            In Boolean          Default True,
        CommitPoint         In Number           Default DefaultCommitPoint
);
```

Examples:

/*Encrypt complete schema SCOTT using DES (default) with a generated key*/
SQL>exec RDC_Encrypt_Object.EncryptData(Null,‘SCOTT’);

/*Encrypt complete table SCOTT.EMP using AES-256 with the key “Hello World”* with Salt*/
SQL>exec RDC_Encrypt_Object.EncryptData(‘Hello World’, ‘SCOTT’, ‘EMP’, NULL,’AES 256-Bit’,true);

/*Encrypt table SCOTT.EMP using AES256 with no salt, issuing a commit every 64K records*/
SQL>exec RDC_Encrypt_Object.EncryptData(Null, ‘SCOTT’,’EMP’,NULL,’AES 256-Bit’, False, 64000)

/*Encrypt column SCOTT.EMP.ENAME and show encrypted column count before and after*/
SQL>Begin
dbms_output.put_line(‘Before: ‘||RDC_Encrypt_Util.CountEncryptedColumns);
RDC_Encrypt_Object.EncryptData(Null,‘SCOTT’,’EMP’, 'ENAME');
dbms_output.put_line(‘After: ‘||RDC_Encrypt_Util.CountEncryptedColumns);
End;

2. DecryptData

DecryptData is the core method utilized to *decrypt* your encrypted table data and update it to its unencrypted state. As with encryption, DecryptData can operate on a schema, table, or a given column. You can also specify a commit-point as with the EncryptData method. If you are using an *administrative password* you will need to call the method *login* before calling DecryptData.

The method *CountEncryptedColumns* should be called before and after a call to this method to monitor the current state of encryption. DecryptData only returns a message if there is a fatal error.

Specification:

```
Procedure DecryptData
(
        TableOwner          In Varchar2          Default Null,
        TableName           In Varchar2          Default Null,
        ColumnName          In Varchar2          Default Null,
        CommitPoint         In Number            Default Null
);
```

Examples:

```
/*Decrypt complete schema SCOTT*/
SQL>exec RDC_Encrypt_Object.DecryptData('SCOTT');

/*Decrypt complete table SCOTT.EMP*/
SQL>exec RDC_Encrypt_Object.DecryptData('SCOTT', 'EMP');

/*Decrypt complete table SCOTT.EMP while issuing commit every 2048 records*/
SQL>exec RDC_Encrypt_Object.DecryptData('SCOTT','EMP',NULL, 2048);

/*Decrypt complete table SCOTT.EMP and print encrypted column count before and after decryption call*/
SQL>Begin
dbms_output.put_line('Before: '||RDC_Encrypt_Util.CountEncryptedColumns);
RDC_Encrypt_Object.DecryptData('SCOTT','EMP');
dbms_output.put_line('After: '||RDC_Encrypt_Util.CountEncryptedColumns);
End;
/
```

3. CreateDecryptedView

This method is used to create or recreate a decrypted view against a single base table. Applications should use decrypted views for transparent data encryption. Typically base tables are renamed or synonyms changed, so that applications read and write data to decrypted views and not the underlying encrypted tables.

CreateDecryptedView can be used either to create a new view against an encrypted base table or to modify an existing decrypted view name and/or owner. An encrypted base table can have only one decrypted view at a given point in time.

If the UpdateFlag is set to FALSE, your decrypted view will be a read-only view. The default value for UpdateFlag is TRUE, which allows DDL against the decrypted view as if it was a table.

For updateable decrypted views (UpdateFlag=TRUE) the Encryption Wizard creates an instead-of-trigger to process SQL update, insert and delete statements. If there is a primary key on the table being decrypted, the instead-of-trigger will use this key to process DDL; otherwise a rowid column will be added to the decrypted view.

To prevent CreateDecryptedView from using primary primary key values, update the column *use_rowid_for_views* to 'X in the table *rdc_encrypt_license*. In this case, a rowid column will be added to the decrypted view regardless of the existence of a primary key.

Text of the decrypted view trigger can be found in the table *encrypted_table,* stored in the column *trigger_text*.

Specification:

```
Procedure CreateDecryptedView
(
        TableOwner                  In Varchar2,
        TableName                   In Varchar2,
        ViewOwner                   In Varchar2         Default Null,
        ViewName                    In Varchar2         Default Null,
        UpdateFlag                  In Boolean          Default True,
        OverWrite                   In Boolean          Default True
);
```

Examples

/*Creates an updateable decrypted view named V_EMP in the schema SCOTT*/
SQL>exec RDC_Encrypt_Object.CreateDecryptedView('SCOTT', 'EMP','SCOTT','V_EMP');

/*Creates a read-only decrypted view named V_EMP in the schema SCOTT*/
SQL>exec RDC_Encrypt_Object.CreateDecryptedView('SCOTT', 'EMP','SCOTT','V_EMP', False);

4. DropDecryptedView

DropDecryptedView is used to drop an existing updateable or read-only decrypted view created by the Encryption Wizard. This method should always be called after you decrypt a table, since the remaining decrypted view will be invalid after your table data is returned to its original state.

To drop a decrypted view, you do not need to remember the view name, simply the base table that it was built against. To find a View Name for a given encrypted base table, query the table *encrypted_table*.

Specification

```
Procedure DropDecryptedView
(
        TableOwner              In Varchar2,
        TableName               In Varchar2,
        OverWriteFlag           In Boolean      Default True
);
```

Examples

/* drops any decrypted view associated with the base table SCOTT.EMP */
SQL>exec RDC_Encrypt_Object.DropDecryptedView('SCOTT', 'EMP');

5. SetDefaultPassword

This method is used to assign a *Default Runtime Password* for a database object as described in the Encryption Wizard User Manual. A user has access to decryption and encryption operations based on their Oracle database grants and roles. If a Default Runtime Password is assigned, then an end-user must authenticate this password at runtime through a package call to *rdc_encrypt_runtime.login* as discussed in the next chapter.

A Default Runtime Password can be assigned for a complete schema, table, or column. To remove a runtime password, *SetDefaultPassword is* called with a null password. RDC_Encrypt_User authenticates through the administration password and not this method. The Default Runtime Password can be over-ridden by a *User-Specific Runtime Password* assigned to a given user through a method call to SetRestrictedPassword.

Specification

```
Procedure SetDefaultPassword
(
        Password                In Varchar2,
        TableOwner              In Varchar2,
        TableName               In Varchar2     Default Null,
        ColumnName              In Varchar2     Default Null,
        NoCommit                In Boolean      Default False
);
```

Examples

```
/*Requires login to decrypt/encrypt objects in the schema SCOTT. The password is TIGER*/
SQL>exec RDC_Encrypt_Object.SetDefaultPassword('TIGER', 'SCOTT');

/*Removes the above Runtime Password*/
SQL>exec RDC_Encrypt_Object.SetDefaultPassword(NULL,'SCOTT');

/*Resets the above Runtime Password, but only requires it for a single column*/
SQL>exec RDC_Encrypt_Object.SetDefaultPassword('TIGER','SCOTT','EMP','SAL');
```

6. InsertRestrictedUser

This method is used to create or modify a *Restricted User List* as described in the Encryption Wizard User Manual. By default a user has access to decryption and encryption operations based on their Oracle database grants and roles.

Restricted User Lists grant users access to a given encrypted schema, table, or column and blocks *all* other users not listed. Once all users are deleted from a Restricted User List, the objects of that list are not restricted and again can be decrypted or encrypted by any Oracle user with access to the underlying database object.

The three valid *authorization types* that are granted to a user for a given database object are full privileges (read/write) ‘E’, encryption (write-only) 'e', or decryption privileges (read-only) ‘D’. Write-only authorizations, 'e', cannot be used in conjunction with decrypted views.

Restricted users and their authorizations can be queried directly from the table *encrypted_column_user*.

Specification

```
Procedure InsertRestrictedUser
(
        TableOwner                      In Varchar2,
        TableName                       In Varchar2,
        ColumnName                      In Varchar2,
        UserName                        In Varchar2,
        AuthorizationType               In Char,
        NoCommit                        In Boolean Default False
);
```

Examples

/*Restricts access of SCOTT.EMP to the user SYSTEM*/
SQL>exec RDC_Encrypt_Object.InsertRestrictedUser(‘SCOTT’,’EMP’,NULL,’SYSTEM’,’E');

/*Restricts access of SCOTT.EMP.SAL to SYSTEM with only decryption (Read) privileges*/
SQL>exec RDC_Encrypt_Object.InsertRestrictedUser(‘SCOTT’,’EMP’,SAL,’SYSTEM’,’D’);

/*Restricts access of SCOTT.EMP.SAL to SYSTEM with only encryption (Write) privileges*/
SQL>exec RDC_Encrypt_Object.InsertRestrictedUser(‘SCOTT’,’EMP’,BONUS','SYSTEM’,’e’);

7. SetRestrictedPassword

This method is used to assign a *User-Specific Runtime Password* to a user defined in a *Restricted User List* as described in the Encryption Wizard User Manual. By default a user has access to decryption and encryption operations based on their Oracle database grants and roles if they appear on a restricted user list. If a *Runtime Password* is assigned, then an end-user must authenticate this password at runtime through the package call *rdc_encrypt_runtime.login* as discussed in the next chapter.

To remove a runtime password, SetRestrictedPassword must be called with a null password. Of course, once a User or *Restricted User List* is removed, the runtime password is also deleted. Only users on *Restricted User Lists* may be assigned runtime passwords.

A User-Specific Runtime Password may be assigned to a given user for a complete schema, a table, or a single column depending on the scope defined in this method call. SetRestrictedPassword will override any *Default Runtime Password* assigned to a given schema, table, or column – for the user specified in the method call. RDC_Encrypt_User authenticates through the administration password and not this method.

Specification

```
Procedure SetRestrictedPassword
(
        UserName                In Varchar2,
        Password                In Varchar2,
        TableOwner              In Varchar2,
        TableName               In Varchar2     Default Null,
        ColumnName              In Varchar2     Default Null,
        NoCommit                In Boolean      Default False
)
```

Examples

/*Requires SYSTEM to login to decrypt/encrypt the schema scott, with the password TIGER*/
SQL>exec RDC_Encrypt_Object.SetRestrictedPassword('SYSTEM','TIGER','SCOTT');

/*Removes the above Runtime Password*/
SQL>exec RDC_Encrypt_Object.SetRestrictedPassword('SYSTEM',NULL,'SCOTT');

/*Resets the above Runtime Password, but only requires it for a single column*/
SQL>exec RDC_Encrypt_Object.SetRestrictedPassword('SYSTEM','TIGER','SCOTT','EMP','SAL');

8. DeleteRestrictedUser

DeleteRestrictedUser is used to remove either one or all users from a Restricted User List. This operation can be performed at the Schema, Table, or Column level. If a Restricted User List is removed completely by specifying a NULL Username or if the last user is removed from a Restricted User List, access to the given object will be granted based only on database privileges.

Specification

```
Procedure DeleteRestrictedUser
(
        TableOwner                      In Varchar2,
        TableName                       In Varchar2     Default Null,
        ColumnName                      In Varchar2     Default Null,
        UserName                        In Varchar2     Default Null,
        NoCommit                        In Boolean      Default False
);
```

Examples

/*removes SYSTEM from the restricted user list for table SCOTT.EMP*/
SQL>exec RDC_Encrypt_Object.DeleteRestrictedUser('SCOTT','EMP',NULL,'SYSTEM');

/*removes restriction of SCOTT.EMP.SAL from SYSTEM */
SQL>exec RDC_Encrypt_Object.DeleteRestrictedUser('SCOTT','EMP','SAL','SYSTEM');

/*removes all restricted lists from the schema SCOTT, thus allowing users to access encrypted data*/
SQL>exec RDC_Encrypt_Object.DeleteRestrictedUser('SCOTT');

9. SetAudting

This method allows for session auditing to be specified for any encrypted schema, table, or column. Auditing can be specified for actions of Encryption (E), Decryption (D), or both (B). All session auditing records are inserted into the table *encrypt_audit.*

Specification

```
Procedure SetAuditing
(
        TableOwner                  In Varchar2,
        TableName                   In Varchar2,
        ColumnName                  In Varchar2,
        AuditType                   In Varchar2
);
```

/*Set session auditing to encryption (write) for all encrypted tables within the schema scott */
SQL>RDC_Encrypt_Util.SetAuditing('SCOTT', NULL, NULL, 'E');

/*Set session auditing for all operations against the table SCOTT.EMP*/
SQL>RDC_Encrypt_Util.SetAuditing('SCOTT','EMP',NULL,'B');

/*Set session auditing for Decryption (read) for column EMP.SAL*/
SQL>RDC_Encrypt_Util.SetAuditing('SCOTT','EMP',SAL,'D');

10.DeleteAuditing

To turn-off auditing for a given schema, table, or column we call the method DeleteAuditing. This method returns an integer representing the number of columns successfully removed from auditing as a result of the function call.

Specification

```
Function DeleteAuditing
(
        TableOwner                  In Varchar2,
        TableName                   In Varchar2,
        ColumnName                  In Varchar2
)
Return Number;
```

Examples

/*turns off auditing for all objects in the schema scott*/
SQL>exec dbms_output.put_line(RDC_Encrypt_Util.deleteauditing('SCOTT'));

/*turns off auditing for the table SCOTT.EMP*/
SQL>exec dbms_output.put_line(RDC_Encrypt_Util.deleteauditing('SCOTT', 'EMP'));

11.SetAdminPassword

The Administrative Password is used to prevent any Oracle user, even a DBA, from unauthorized use of the Encryption Wizard. Once this password is set, users will need to use the *.login* method before using any of the administrative methods. Runtime decryption and encryption does not require this password, only administrative functions.

To set the administrative password, simply pass in a string to SetAdminPassword. If you are already using an *administrative password* you will need to call the method *login* before calling SetAdminPassword.

Specification

```
Procedure SetAdminPassword
(
        Password        In Varchar2
);
```

Example

/* sets the Administrative Password to 'My Password */

SQL>exec RDC_Encrypt_Object.SetAdminPassword('My Password');

/* Logs on with the with the password and changes the Administrative Password to 'New Password' */

SQL>exec RDC_Encrypt_Object.Login('My Password');
SQL>exec RDC_Encrypt_Object.SetAdminPassword('New Password');

12.Login

Login is used to identify Encryption Wizard Administrators if there is an Administrative Password set. The Login method must be used to authenticate an Encryption Wizard Administrator before any operations on existing encrypted data, decrypted views, audit logs, runtime passwords, or restricted user lists can occur. After three attempts to Login, this method requires you to reconnect to Oracle.

The Login method can only be called by connecting to the Oracle account *rdc_encrypt_user*. All other users authenticate using Runtime Passwords and by evoking the rdc_encrypt_runtime.login method as discussed in the next chapter.

Specification

```
Procedure Login
(
        Password        In Varchar2
);
```

Example

/*logs in as administrator with the password “Hello World”*/

SQL>exec RDC_Encrypt_Object.login(‘Hello World’);

13. ChangeRollbackSegment

This method allows you to specify a rollback segment for your session. Simply call this method with a valid online rollback segment name and the new rollback segment will be set as the default for the remainder of the session.

Specification

```
Procedure ChangeRollbackSegment
(
        SegmentName  In Varchar2
);
```

Example

SQL>exec RDC_Encrypt_Util.ChangeRollbackSegment('Roll_Large');

14.BackupKeys

This method is used to backup the encryption keys of the Encryption Wizard to a flat-file. Backupkeys can backup the keys for a database, schema, table, or single column. You may also supply an optional password to insure against unauthorized restoration of encryption keys.

Key backups are only valid for the database that they are backed-up from. This insures that a hacker cannot restore these keys using the Encryption Wizard if they are ever stolen. Key backups may also not span different releases of Oracle.

Specification

```
Procedure BackupKeys
(
        FileName                        In Varchar2,
        TableOwner                      In Varchar2     Default Null,
        TableName                       In Varchar2     Default Null,
        ColumnName                      In Varchar2     Default Null,
        Password                        In Varchar2     Default Null
);
```

Examples

/* backs up all encryption keys in the database to a flat file */
SQL>exec rdc_encrypt_backup.BackupKeys('/u01/encrypt/bak/myfile.txt');

/* backups up the encryption keys for the schema SCOTT using the password "TIGER" */
SQL>exec rdc_encrypt_backup.BackupKeys('/01/temp.txt, 'SCOTT', Null, Null, 'TIGER');

15.RestoreKeys

This method is used to restore the encryption keys of a flat-file to the Encryption Wizard schema. RestoreKeys can restore the keys for a database, schema, table, or single column. You may also supply an optional password to insure against unauthorized restoration of encryption keys.

If you provide the Boolean TRUE for the OverWriteFlag, the Encryption Wizard will overwrite any existing keys in the database that are restored from the flat-file.

Specification

```
Procedure RestoreKeys
(
        FileName                        In Varchar2,
        TableOwner                      In Varchar2     Default Null,
        TableName                       In Varchar2     Default Null,
        ColumnName                      In Varchar2     Default Null,
        Password                        In Varchar2     Default Null,
        OverWriteFlag                   In Boolean      Default False
);
```

Examples

/*Restores the encryption keys for an entire database*/
SQL>exec rdc_encrypt_backup.RestoreKeys('/u01/encrypt/bak/myfile.txt');

/*Restores the Encryption Keys for the table EMP, overwriting any existing keys*/
SQL>exec rdc_encrypt_backup.RestoreKeys('/u01/tmp.txt, 'SCOTT', 'EMP', Null, Null, True);

16. RestoreColumn

This method is used to restore the encryption key of a column that was previously encrypted. After deletion, every column key is sent to a history table in case the key is needed later in a data recovery scenario.

For instance, an old database backup may contain an encrypted table that is no longer used in production. *RestoreColumn* will try to retrieve key data from the Encryption Wizard history tables: *encrypted_column_history* and *encrypted_table_history*.

RestoreColumn does not restore *restricted user lists* or *runtime passwords* for a column, so be sure to reset any security settings after a restore operation is successful. For normal key backup and recovery operations use *BackupKeys* and *RestoreKeys*.

Specification

```
Procedure RestoreColumn
(
        TableOwner              In Varchar2,
        TableName               In Varchar2,
        ColumnName              In Varchar2
)
```

Examples

/*Restores the encryption key for the column ename in the table scott.emp */
SQL>exec rdc_encrypt_object.restorecolumn('SCOTT', 'EMP', 'ENAME');

17. SetDefaultMask

This method is used in conjunction with restricted user lists and runtime passwords. By default, if a user fails authentication with the Encryption Wizard, an Oracle error will occur if a decryption (read) is attempted. These Oracle errors are sometimes confusing for applications and not necessary.

With a default mask for a given column, an unauthenticated user simply will see a null or constant string such as '???' in place of the data:

Select Ename From *Decrypted_View*.

ENAME

???
???
???

When using *SetDefaultMask*, make sure unauthenticated users and scripts are not selecting from decrypted data and using default masks to change data or make business decisions. Encryption Wizard auditing is not performed when a unauthenticated user selects a default mask, since no decryption attempt is made.

Specification

```
Procedure SetDefaultMask
(
        DefaultMask     In Varchar2,
        TableOwner      In Varchar2,
        TableName       In Varchar2,
        ColumnName      In Varchar2
)
```

Examples

```
/*Sets the default mask to ??? for  the column ename in the table scott.emp */
SQL>exec RDC_Encrypt_Object.SetDefaultMask('???', 'SCOTT', 'EMP', 'ENAME');

/*Sets the default mask to return a NULL value  for  the column ename in the table scott.emp */
SQL>exec RDC_Encrypt_Object.SetDefaultMask(Null, 'SCOTT', 'EMP', 'ENAME');

/*Sets the default mask to  0 for the number column sal in the table scott.emp */
SQL>exec RDC_Encrypt_Object.SetDefaultMask(0, 'SCOTT', 'EMP', 'SAL');

/*Sets the default mask to return SYSDATE for the date column hiredate in the table scott.emp */
SQL>exec RDC_Encrypt_Object.SetDefaultMask('SYSDATE', 'SCOTT', 'EMP', 'HIREDATE');
```

18. GetDefaultMask

GetDefaultMask returns the current mask for a given encrypted column. If the default mask is set to NULL, this function will return NULL. If there is no default mask set for the given column, then *GetDefaultMask* will raise an Oracle exception. To avoid the exception, use *IsDefaultMask* in conjunction with this function as shown in the example below.

Regardless, of the original data type of the mask, *GetDefaultMask* returns a Varchar2 representation.

Specification

```
Procedure GetDefaultMask
(
        TableOwner      In Varchar2,
        TableName       In Varchar2,
        ColumnName      In Varchar2
)
```

Examples

```
/*Prints the default mask for the encrypted column ename*/
SQL>exec  bms_output.put_line(RDC_Encrypt_Object.GetDefaultMask('SCOTT', 'EMP', 'ENAME'));

/* Prints the default mask for the encrypted column ename, if one exists  */
Begin
        If RDC_Encrypt_Object.IsDefaultMask('SCOTT', 'EMP', 'ENAME') Then
          dbms_output.put_line(RDC_Encrypt_Object.GetDefaultMask('SCOTT', 'EMP', 'ENAME'));
        Else
          dbms_output.put_line('No default mask found for ename');
        End If;
End;
/
```

19. RemoveDefaultMask

This method is used remove default mask(s) for any schema, table or column that were set using the API call *SetDefaultMask.* After a default mask is removed, by default again an unauthenticated user will receive an Oracle error when attempting a decryption operation with the Encryption Wizard.

Specification

```
Procedure RemoveDefaultMask
(
        TableOwner      In Varchar2             Default Null,
        TableName       In Varchar2             Default Null,
        ColumnName      In Varchar2             Default Null
)
```

Examples

```
/*Removes any default mask for all encrypted database columns  */
SQL>exec RDC_Encrypt_Object.RemoveDefaultMask;

/*Removes any default masks for all encrypted columns in the schema scott */
SQL>exec RDC_Encrypt_Object.RemoveDefaultMask('SCOTT');

/*Removes any default masks for encrypted columns in the table scott.emp  */
SQL>exec RDC_Encrypt_Object.RemoveDefaultMask("SCOTT', 'EMP');

/*Removes the default mask for the column hiredate, if one exists */
SQL>exec RDC_Encrypt_Object.RemoveDefaultMask( 'SCOTT', 'EMP', 'HIREDATE');
```

20. CountEncryptedColumns

This Method is used in determining if a given column is encrypted. It can also be used to check to see if a given table or schema has encrypted columns within it. The method CountEncryptedColumns will either return a positive integer specifying the number of encrypted columns for the object scope, or the function will return 0 if there are no currently encrypted columns.

Specification

```
Function CountEncryptedColumns
(
        TableOwner      In Varchar2     Default Null,
        TableName       In Varchar2     Default Null,
        ColumnName      In Varchar2     Default Null
)
Return Number;
```

Example

```
/*return the number of columns encrypted in the SCOTT schema*/
SQL>exec dbms_output.put_line(RDC_Encrypt_Util.CountEncryptedColumns('SCOTT'));

/*return 1 if EMP.SAL is encrypted, 0 if it is not */
SQL>exec dbms_output.put_line(RDC_Encrypt_Util.CountEncryptedColumns('SCOTT','EMP','SAL'));

/*Pseudo-Code for EMP.SAL check within 3 GL Language via generic CALL method*/
If ( call((RDC_Encrypt_Util.CountEncryptedColumns('SCOTT','EMP','SAL')) > 0)
        print('column encrypted');
```

21. CountAuditLog

This method counts the number of audit records in the table *encrypt_audit* for a given schema, table, or column. Records no longer needed from this table can be deleted at any time manually.

```
Function CountAuditLog
(
        TableOwner      In Varchar2     Default Null,
        TableName       In Varchar2     Default Null,
        ColumnName      In Varchar2     Default Null
)
Return Number;
```

/* Counts all audit records for the schema SCOTT */

SQL>exec dbms_output.put_line(RDC_Encrypt_Util.CountAuditLog('SCOTT'));

/* Counts all audit records for the column EMP.SAL */

SQL> exec dbms_output.put_line(RDC_Encrypt_Util.CountAuditLog ('SCOTT','EMP','SAL'));

/*Counts all Encryption Wizard audit records in your RDBMS*/

SQL> exec dbms_output.put_line(RDC_Encrypt_Util.CountAuditLog;

22. ReKeyData

This method allows one to *re-key* the encryption keys for a given schema, table, or column. Make sure database users are not reading and/or writing to the encrypted columns chosen for re-keying. This process will take approximately as long as the method EncryptData on any given data set.

New and old key values are stored in the table internal_audit. In the case of a re-keying failure, such as a database shutdown or lock contention error, simply call the method again and re-key recovery will commence for the remaining rows that were not re-keyed.

```
Procedure RekeyData
(
        KeyValue                In Varchar2     Default Null,
        TableOwner              In Varchar2,
        TableName               In Varchar2     Default Null,
        ColumnName              In Varchar2     Default Null,
        CommitPoint             In Number       Default Null
);
```

/* Re-keys all encrypted columns for the schema SCOTT */

SQL>exec RDC_Encrypt_Object.ReKeyData(Null, ‘SCOTT’);

/* Re-keys all encrypted rows for the table SCOTT.EMP */

SQL>exec RDC_Encrypt_Object.ReKeyData(Null, ‘SCOTT’, 'EMP');

/*Re-keys the column SAL using a specified string as the seed to the new key */

SQL>exec RDC_Encrypt_Object.ReKeyData('Hello World', ‘SCOTT’, 'EMP', 'SAL');

23. IsObjectAudited

This method returns TRUE if a given schema, table, or column contains columns being audited, and FALSE otherwise.

Specification

```
Function IsObjectAudited
(
        TableOwner      In Varchar2,
        TableName       In Varchar2     Default Null,
        ColumnName      In Varchar2     Default Null
)
Return Boolean;
```

Examples

/*PL/SQL Example – Print Notification if SCOTT.EMP contains no audited columns*/

```
Begin
        If RDC_Encrypt_Util.IsObjectAudited('SCOTT','EMP') = False Then
                DBMS_Output.Put_Line('Encrypted Table Scott is not being audited');
        End If;
End;
/
```

24. IsDecryptedView

IsDecryptedView is a function that returns TRUE if a decrypted view has been created against a given base table and returns FALSE if there is no such decrypted view created for the encrypted base table.

Specification

```
Function IsDecryptedView
(
        TableOwner                      In Varchar2,
        TableName                       In Varchar2
)
Return Boolean;
```

Examples

```
/*PL/SQL Example - creates a decrypted view V_EMP for table SCOTT.EMP if there is no such view*/
Begin
        If RDC_Encrypt_Util.IsDecryptedView('SCOTT','EMP') = False Then

        RDC_Encrypt_Util.CreateDecryptedView('SCOTT','EMP','SCOTT','V_EMP');
        End If;
End;
/
```

25. EncryptNullData

The EncryptNullData method is used to set null encryption globally for the Encryption Wizard. Usually the Encryption Wizard ignores null values, but there are some instances in which even null values may need to be encrypted. Use this function to turn on and off NULL value encryption.

Setting this value will not affect null values currently encrypted or not encrypted. To change their value, set them to NULL *after* the call to EncryptNullData using a decrypted view or a direct call to the runtime library as such:

Update V_Emp Set COMM = Null Where COMM Is Null;

In this case, all of the null values contained in the decrypted view V_EMP for the column EMP.COMM will now be encrypted – If EncryptNullData is set to YES for the database as whole or for the object EMP.COMM.

For databases with a great deal of null data, encrypting this data will require more free space within a given tablespace to store the encrypted values.

Specification

```
Procedure EncryptNullData
(
        Val                 In Varchar2,
        TableOwner          In Varchar2     Default Null,
        TableName           In Varchar2     Default Null,
        ColumnName          In Varchar2     Default Null
)
```

Examples

/* Instructs the Encryption Wizard to encrypt null data from this point forward */
Exec RDC_Encrypt_Util.EncryptNullData('Yes');

/* Instructs the Encryption Wizard to ignore null data from this point forward */
Exec RDC_Encrypt_Util.EncryptNullData('No');

/* Instructs the Encryption Wizard to encrypt null data for the column COMM */
Exec RDC_Encrypt_Util.EncryptNullData('Yes', 'SCOTT', 'EMP', 'COMM');

/* Instructs the Encryption Wizard to not encrypt null data for the schema SCOTT */
Exec RDC_Encrypt_Util.EncryptNullData('No', 'SCOTT');

26. IsRecovery

The IsRecovery method method returns a Boolean if recovery is needed on an encrypted base table. Recovery implies that an attempt to decrypt or encrypt an entire base table was only partially successful.

Specification

```
Function IsRecovery
(
        TableOwner      In Varchar2,
        TableName       In Varchar2
)
Return Boolean;
```

Examples

/*PL/SQL example – If SCOTT.EMP needs recover, then decrypt the table*/

```
Begin
        If RDC_Encrypt_Util.IsRecovery('SCOTT','EMP') = True Then
                        RDC_Encrypt_Object.DecryptData('SCOTT','EMP');
        End If;
End;
/
```

27. GetVersionNo

GetVersionNo returns the current string of the Encryption Wizard version source code that you are running. This may be helpful in troubleshooting.

Specification

```
Function GetVersionNo
Return Varchar2
```

Examples

SQL>exec dbms_output.put_line(RDC_Encrypt_Object.getversionno);

28. GetLastErrorMsg

GetLastErrorMsg Method is used to retrieve the last Encryption Wizard error received for a given session. After a call to this method, this last error message is again initialized to NULL. All Encryption Wizard errors are recorded in the table *Internal_Error*.

Specification

```
Function GetLastErrorMsg
Return Varchar2;
```

Examples

```
/*Prints the last error for your session*/

SQL>exec dbms_output.put_line('Encryption Wizard Message: '||RDC_Core.GetLastErrorMsg);

/* Performs error checking after an attempt to encrypt the table SCOTT.EMP*/
SQL>
Begin
        RDC_Encrypt_Object.EncryptData(Null, 'SCOTT', 'EMP');

        Exception When Others Then
                DBMS_Output.Put_Line('Err: '||Nvl(RDC_Core.GetLastErrorMsg, SQLERRM) );
End;
/
```

Section Two

Runtime Methods

The Encryption Wizard Runtime Methods are designed for use by any end-user who is required to read and/or write encrypted data. The Runtime Library consists of two core functions, EncryptData and DecryptData. These functions are overloaded to return Varchar2, Blob, Clob, Date, and Number values depending on the type of parameter passed into the method. Usually these functions are hidden from the user through the implementation of *Decrypted Views*.

To use both EncryptData and the DecryptData function a user needs to pass in the integer value of the *column_seq* for that column. This value can be queried from the view encrypted_tab_columns or this value can be returned from the function *GetColumnSeq* as discussed in the previous section. The method *GetColumnSeq* is called from RDC_Encrypt_Runtime and can be accessed by the general Oracle user.

Along with the core encryption/decryption methods of the Runtime Library, a login method is also provided for end-users to optionally authenticate *Runtime Passwords*. The default configuration of the runtime library allows all Oracle users the ability to encrypt or decrypt data *dependent on their base table grants*.

Because DBA-level users have access to all database objects, to restrict a DBA's access to encrypted data, the Encryption Wizard allows you to define *Default Runtime Passwords* which require all users to authenticate themselves against encrypted objects. You may also employ *Restricted User Lists* and *User-Specific Runtime Passwords* to lock out users such as SYS or SYSTEM completely.

To add an additional layer of protection against data-theft by non-DBA users who may have the proper Oracle grants - you may issue this command to revoke the Encryption Wizard's Runtime Package from public.

```
SQL>revoke execute on rdc_encrypt_runtime from public;
```

After this command succeeds, simply grant privileges to execute the Runtime Library to individual users as such:

```
SQL>grant execute on rdc_encrypt_runtime to SCOTT;
```

1. EncryptData

EncryptData takes any valid string, number, or date and encrypts the data using the key specified by the *Column_Seq*, which is the primary key of the table encrypted_column.

EncryptData can be used within PL/SQL, as a user-defined function, or embedded within a language such as Java, C++ or PHP. For Columns that specify DES or AES encryption, the Varchar2, BLOB, or CLOB value will always be returned as rounded upwards to a multiple of 8 bytes if necessary. Use *EncyptDataNLS* for NVarchar2 and NCHAR datatypes.

Specification

```
Function EncryptData
(
        ColumnSeq                       In Number,
        InputData                       In Varchar2, Nvarchar2, Date, Number, Blob, Clob*
)
Return Varchar2, Nvarchar2, Date, Number, CLOB, BLOB*
```

*Overloaded

Examples

```
/* Directly Inserts a new employee with an encrypted ENAME to the base table SCOTT.EMP */
SQL>Insert Into Emp
(
        EmpNo,
        Ename
)
Values
(
        23454,
        RDC_Encrypt_Runtime.EncryptData( ColumnSeq, 'Sarah Jones')
);

/*Updates a new encrypted Salary number directly to the encrypted base table SCOTT.EMP*/
SQL>Update EMP Set SAL = RDC_Encrypt_Runtime.EncryptData(ColumnSeq, 65000)
Where RDC_Encrypt_Runtime.DecryptData(ColumnSeq, Ename) = 'Sarah Jones';

/* Updates SCOTT.EMP directly by encrypting the value "Joe Smith" for the column ENAME.
 Note the use of GetColumnSeq to determine the primary key for EMP.ENAME*/

SQL>Update SCOTT.EMP Set Ename =
rdc_encrypt_runtime.EncryptData
(
        RDC_Encrypt_Runtime.GetColumnSeq('SCOTT','EMP','ENAME'),
        'Joe Smith'
) Where EMP_NO = &Emp_No;
```

2. DecryptData

DecryptData takes any valid string, number, or date and decrypts the data using the password key specified by the *ColumnSeq* variable, thus the key relevant for a given database column. DecryptData can be used within PL/SQL, as a user-defined function, or embedded within a language such as Java, C++ or PHP.

For Columns that specify DES encryption the Varchar2 input variable must be a multiple of 8 bytes or DecyptData will issue a fatal-error. Use *DecryptDataNLS* for NVarchar2 and NCHAR datatypes.

Specification

```
Function DecryptData
(
        ColumnSeq                                   In Number,
        InputData                                   In Varchar2, Nvarchar2, Date, Number, Blob, Clob*
)
Return Varchar2, Nvarchar2,  Date, Number, Blob, Clob*
```

*Overloaded

Examples

/*Select un-encrypted salary from the table EMP*/

SQL>select rdc_encrypt_runtime.DecryptData(*ColumnSeq,* SAL) from EMP;

/*Select the count from the encrypted table EMP where the salary is greater than 50,000*/

SQL>select count(*) from EMP where rdc_encrypt_runtime.DecryptData(*ColumnSeq,* SAL) > 50000

/*Decrypts the ENAME column using a nested call to GetColumnSeq*/

```
SQL>Select
rdc_encrypt_runtime.DecryptData
(
        RDC_Encrypt_Runtime.GetColumnSeq('SCOTT','EMP','ENAME'),
        ENAME
)
From
SCOTT.EMP;
```

3. GetColumnSeq

The *GetColumnSeq* method is used to retrieve the primary key, *column seq,* from the table encrypted_column. This method returns a positive integer if the given column is currently encrypted and a NULL value if the column is not encrypted. You can also query the *column seq* directly from the view *encrypted_tab_columns.*

Specification

```
Function GetColumnSeq
(
        TableOwner      In Varchar2,
        TableName       In Varchar2,
        ColumnName      In Varchar2
)
Return Number;
```

Examples

/* Retrieves the column seq for the column EMP.ENAME*/

```
SQL>exec dbms_output.put_line(RDC_Encrypt_Runtime.GetColumnSeq('SCOTT','EMP','ENAME'));
```

/*Decrypts the encrypted ENAME column using a nested call to GetColumnSeq (see next section) */

```
SQL>Select
rdc_encrypt_runtime.DecryptData
(
        RDC_Encrypt_Runtime.GetColumnSeq('SCOTT','EMP','ENAME'),
        ENAME
)
From
SCOTT.EMP;
```

4. Login

The Login Method is used to authenticate users against objects that are protected with Runtime Passwords. This method validates the Runtime Password assigned to protect a given schema, table, or column. After a user has been authenticated with the correct password, they are allowed their specified access to encrypted objects for the remainder of their Oracle session. If a user reconnects they will need to again call this method to authenticate themselves against any password-protected objects.

Login can be called directly from SQL*Plus or embedded in a language such as Java, C, or PHP. Login is always called by oracle accounts other than the Encryption Wizard account.

Specification

```
Procedure Login
(
        Password                In Varchar2             Default Null,
        TableOwner              In Varchar2             Default Null,
        TableName               In Varchar2             Default Null,
        ColumnName              In Varchar2             Default Null
);
```

Examples

/*Authenticates the current Oracle user for the Table EMP*/

SQL>exec rdc_encrypt_runtime.login('MyPassword','SCOTT','EMP');

/* Authenaticates the current Oracle user for the Schema SCOTT*/

SQL>exec rdc_encrypt_runtime.login('MyPassword','SCOTT');

5. GetLastErrorMsg

GetLastErrorMsg Method is used to retrieve the last Encryption Wizard error received for a given session. After the call to this method, this last error message is again initialized to NULL. All Encryption Wizard errors are recorded in the table *Internal_Error.*

Specification

```
Function GetLastErrorMsg
Return Varchar2
```

Examples

/*prints out the last message from the Encryption Wizard for the given session*/

```
SQL>exec dbms_output.put_line('Last Message: '||RDC_Encrypt_Runtime.GetLastErrorMsg);
```

/* Performs error checking after an attempt to encrypt the table SCOTT.EMP*/

```
SQL>
Begin
        RDC_Encrypt_Object.EncryptData(Null, 'SCOTT', 'EMP');

        Exception When Others Then
                DBMS_Output.Put_Line('Error: '||Nvl(RDC_Core.GetLastErrorMsg, SQLERRM) );
End;
/
```

Section Three

HSM Management

The Encryption Wizard HSM Methods allow Encryption Wizard Administrators the ability to view and change parameters on the Encryption Wizard to reflect changes in the HSM configuration.

All parameters for the HSM module are encrypted in the table *hsm_parameters.* Values like the HSM slot number or the HSM password may periodically change, and these changes can easily be propagated to the Encryption Wizard.

All errors for these API calls and messages are stored in the System Log, the table *internal_audit.*

In most cases, if an HSM card is not present or an invalid Master Key is passed to the Encryption Wizard, an “Invalid License” error will appear in the system log – thus protecting your HSM dependent data from corruption. Encrypted data from an HSM master key cannot be modified by the Encryption Wizard unless the returned master key matches an internal HSM hashing value for each column.

1. Initialize

Initialize is the method run by the script rdc_hsm_install.sql. Because initialize resets all the keys and licensed passwords for the Encryption Wizard, this procedure checks for two basic things before running:

1. Have any previous HSM_Parameters table been dropped manually?
2. Are there zero encrypted columns in the database?

Once these two prerequisites are met, initialize will create a key store on the HSM card based on the Key Name provided and the HSM password, then Initialize will recreate the HSM_Parameter table.

Specification

```
Procedure Initialize
(
        CardType                In Varchar2,
        Password                In Varchar2,
        KeyName                 In Varchar2,
        SlotNum                 In Number           Default Null
);
```

Examples

/ Create a new key store on the HSM, searchable for the HSM slot */*
SQL> exec rdc_encrypt_hsm.initialize('Luna PCI 7000','My Password','My Key');

/ Creates a new key store on the HSM that must exist on slot 2 */*
SQL> exec rdc_encrypt_hsm.initialize('Luna PCI 7000','My Password','My Key','2');

/ With Error-Trapping */*

```
SQL>
Begin
        RDC_Encrypt_HSM.Initialize("Luna PCI 7000, 'Hello World', 'My Key'');

        Exception When Others Then
                DBMS_Output.Put_Line('Error: '||Nvl(RDC_Core.GetLastErrorMsg, SQLERRM) );
End;
/
```

2. GetParameter

GetParameter is exclusively used to view the contents of the encrypted table, *HSM_Parameters*. For a list of possible parameters, execute this query:

SQL>Select Name from HSM_Parameters;

Specification

```
Function GetParameter
(
        Name                    In Varchar2,
        Password                In Varchar2
)
Return Varchar2;
```

Examples

/ Displays the current slot number of the HSM card installed */*
SQL> Select RDC_Encrypt_HSM.GetParameter('Slot Num', 'HSM Password') from dual;

/ Displays the Key Name where the Encryption Wizard master key is stored */*
SQL> exec dbms_output.put_line(RDC_Encrypt_HSM.GetParameter('Key Name', 'HSM Password'));

3. GetJavaParameter

GetJavaParameter is primarily used for debugging any errors within the Oracle JVM. It is usually run at the suggestion of the Encryption Wizard support staff

Specification

```
Function GetJavaParameter
(
        Name                    In Varchar2
)
Return Varchar2;
```

Examples

/ Displays the last exception for the Java HSM interface */*
SQL> Select RDC_Encrypt_HSM.GetJavaParameter('LastException') from dual;

/ Displays the key length in bytes of the Master Key stored in the HSM */*
SQL> exec dbms_output.put_line(RDC_Encrypt_HSM.GetJavaParameter('GetKeyLength'));

/ Displays the current Java Library Path of the Oracle RDBMS */*
SQL> Select RDC_Encrypt_HSM.GetJavaParameter('java.library.path') from dual;

4. ChangeParameter

ChangeParameter is used to modify the contents of the highly-encrypted table, HSM_Parameters. For a list of possible parameters, execute this query:

SQL>Select Name from HSM_Parameters;

Specification

```
Procedure ChangeParameter
(
        Name                    In Varchar2,
        Value                   In Varchar2,
        Password                In Varchar2

);
```

Examples

/ Changes the current HSM slot number to 1 */*
SQL> exec RDC_Encrypt_HSM.ChangeParameter('Slot Num', '1', 'HSM Password');

** Changes the current HSM slot number to choose first available, which is -1 */*
SQL> exec RDC_Encrypt_HSM.ChangeParameter('Slot Num', '-1', 'HSM Password');

/ Updates a change to the HSM password for the Encryption Wizard */*
SQL> exec RDC_Encrypt_HSM.ChangeParameter('Password', 'New Password', 'HSM Password');