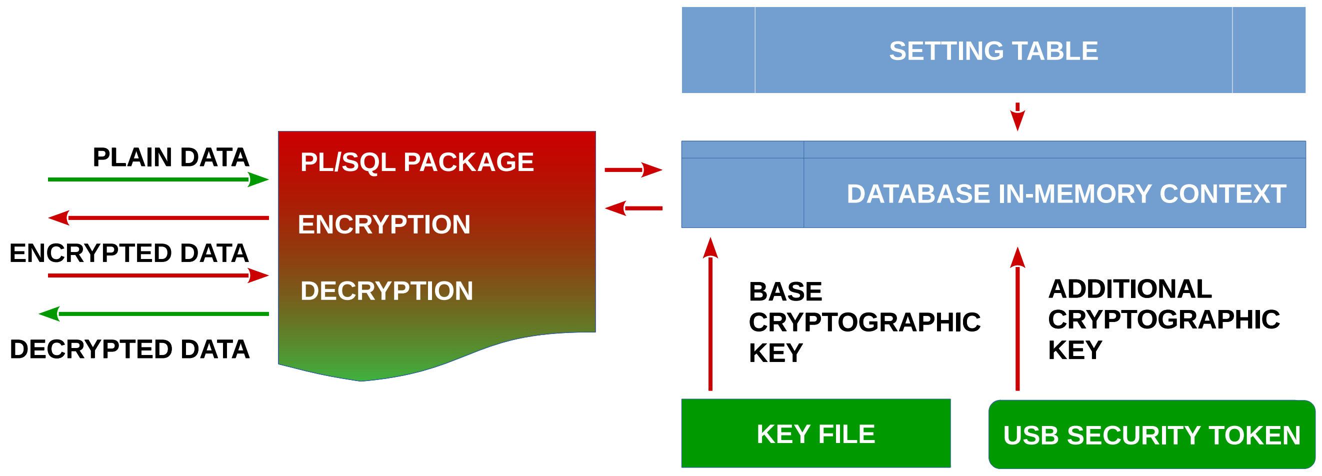
FCBCrypto software UNIX installation and user guide

Table of contents

CONCEPT



Database in-memory context is filled by cryptographic related information from a pre-defined table, an operating system key file and an USB security token (Linux only). Data go through the PL/SQL package to be encrypted or decrypted. Before encryption or decryption actions the PL/SQL package reads cryptographic information from the database in-memory context. Data are encrypted or decrypted and returned back.

BEGINING

FCBCRYPTO software is provided as a binary self-extracting archive file. Its name looks like **fcbcrypto-1.2.40.run** where 1 is a version number, 2 is a subversion number and 40 is a build number. FCBCRYPTO software installation supposes Korn shell[1] (ksh) is installed, valid and available as /usr/bin/ksh[2]. **fcbcrypto-1.2.40.run** file can be run on UNIX like operating systems (IBM AIX, SUN/Oracle Solaris, Linux, HP HP-UX)[3], i.e.

```
$ ./fcbcrypto-1.2.40.run

End User License Agreement
--------------------------

Preface
-------

The following contains specific license terms and conditions for Olexandr
Siroklyn's product(s). By accepting this agreement, you agree to comply with all
the terms and conditions applicable to the specific product(s) included herein.
…

Please type y to accept, n otherwise:
```

As it can be seen from above there is an End User License Agreement you must accept to continue or reject to cancel installation. In case you accept End User License Agreement unpacking starts,

```
Please type y to accept, n otherwise: y
Creating directory fcbcrypto
Verifying archive integrity...  100%   All good.
Uncompressing installer for FCBCRYPTO software  100%
...
```

fcbcrypto directory is created

```
$ ls -ltr
total 1.3M
drwxr-xr-x. 9 **** dba 4.0K Jan 31 18:39 fcbcrypto/
-rwxr-xr-x. 1 **** dba 1.3M Jan 31 18:39 fcbcrypto-1.2.40.run*
```

1 Korn shell is one of prerequisites for Oracle database installation. See Installation and Upgrade Guide for Linux, Installation Guide for IBM AIX, Installation Guide for Oracle Solaris, Installation Guide for HP-UX Itanium
2 if /usr/bin as a catalog for ksh is absolutely unacceptable for you, please either make a symbolic link or change #!/usr/bin/ksh value in fcbcryprto/bin/cre_fcbcrypto.sh file
3 it wasn't tested on HP HP-UX but it should work

```
$ ls -la fcbcrypto
total 44K
drwxr-xr-x. 9 **** dba 4.0K Jan 31 18:39 ./
drwxr-xr-x. 3 **** dba 4.0K Feb  2 10:15 ../
drwxr-xr-x. 2 **** dba 4.0K Jan 31 18:39 bin/
drwxr-xr-x. 2 **** dba 4.0K Jan 31 18:39 dat/
-rw-r--r--. 1 **** dba  851 Jan 31 18:39 .dbvariables
drwxr-xr-x. 2 **** dba 4.0K Jan 31 18:39 doc/
drwxr-xr-x. 2 **** dba 4.0K Jan 31 18:39 log/
drwxr-xr-x. 2 **** dba 4.0K Jan 31 18:39 man/
-rw-r--r--. 1 **** dba 2.2K Jan 31 18:39 .osvariables
drwxr-xr-x. 2 **** dba 4.0K Jan 31 18:39 sql/
drwxr-xr-x. 9 **** dba 4.0K Jan 31 18:39 src/
```

and FCBCRYPTO software installation starts via

```
$ cd fcbcrypto/bin; ./cre_fcbcrypto.sh

[4]Hello!

You are starting installation of FCBCRYPTO software. It's very recommended to do
that under Oracle database binaries' operating system owner account. Your current
account is '****:dba'. You can press Ctrl-C anytime to stop installation. You can
re-run installation via cd fcbcrypto/bin/../bin; ./cre_fcbcrypto.sh call[5]. Before
the next step please make sure following items are filled/setup/working correctly:

  a. fcbcrypto/bin/../.dbvariables file
  b. $ORACLE_HOME environment variable
  c. Oracle listener service
  d. $ORACLE_HOME/bin/sqlplus utility
```

PREREQUISITES

There are prerequisites to continue installation

a) fcbcrypto/bin/../.osvariables file
b) fcbcrypto/bin/../.dbvariables file
c) Oracle database 11g (excepting Express Edition[6]), 12c or 18c
d) $ORACLE_HOME environment variable
e) Oracle listener service
f) $ORACLE_HOME/bin/sqlplus utility
g) database user privileges
h) Oracle Database Java subsystem[7]

a) **.osvariables** file contains a small shell scripts to detect paths to the common UNIX utilities like ls, cat, diff and

4 an example from installation process
5 please re-run exactly by this way “cd fcbcrypto/bin/; ./cre_fcbcrypto.sh”
6 Oracle Database 11g Express Edition
7 a standard database cross-version and cross-edition component or How to Add the JVM Component to an Existing Oracle Database (Doc ID 1461562.1)

so on. Also .osvariables file contains directory aliases for FCBCRYPTO software installation. In common case you don't need to change anything inside this file.

b) **.dbvariables** file contains database related environment variables. Most of their values are wrong for your database case. Some .dbvariables file values are transported to TBL_FCBCRYPTO_SETTING table (see **SETTINGS** chapter) after installation finishes. **You must manually adjust .dbvariables file values relating to you needs and database parameters.**

- **l_EXT_LIB_DIRECTORY**=/tmp points to the operating system directory where external "bridge" C language module will be placed if you're going to use BFILE encryption (see **BFILE ENCRYPTION** chapter). This directory must be readable and writable for the user who performs FCBCRYPTO software installation. This directory must be readable for the Oracle database binaries' owner.

- **l_DB_SERVICE**=orcl sets a database service name to be used in connection process.

- **l_DB_HOST**=localhost sets a host name to be used in connection process.

- **l_DB_PORT**=1521 sets a port number to be used in connection process.

- **l_PACKAGE_OWNER**=system sets a database schema where FCBCRYPTO software will be installed.

- **l_PACKAGE_OWNER_PASS**=system sets a password for the database schema where FCBCRYPTO software will be installed.

- **l_KEYFILE**=key.txt sets a name for the key file where base cryptographic key will be placed.

- **l_KEYFILE_CE**=key_ce.txt sets a name for the key file where core code signing key will be placed.

- 4**l_KEYFILE_DIRECTORY**=DATA_PUMP_DIR sets a database directory (not an operating system directory!) name where **l_KEYFILE** and **l_KEYFILE_CE** will be placed. Please refer to the ALL_DIRECTORIES view for details.

- **l_DB_CONNECTION**=${l_PACKAGE_OWNER}/${l_PACKAGE_OWNER_PASS}@${l_DB_HOST}:${l_DB_PORT}/${l_DB_SERVICE} simply joins together some previous variables to get a database connection string to be used by SQL*Plus utility.

d) **$ORACLE_HOME** environment variable must be setup to provide a correct work of SQL*Plus utility.

e) **Oracle listener service** must be setup and started. Also **l_DB_SERVICE** value should be registered in listener service.

f) $**ORACLE_HOME/bin/sqlplus** utility must be available via **$PATH** environment variable. $**ORACLE_HOME/bin/sqlplus** is used in FCBCRYPTO software installation process to run SQL scripts.

g) A database user where FCBCRYPTO software is going to be installed (see **l_PACKAGE_OWNER** variable from .dbvariables file) must have following grants

- create any context
- drop any context
- create library
- create table
- create procedure
- create sequence
- create trigger

- create type
- create session
- alter session
- select any dictionary
- read/write on **l_KEYFILE_DIRECTORY**4 (see .dbvariables file) directory
- execute on **SYS.DBMS_CRYPTO**[8]
- execute on **SYS.UTL_COMPRESS**[9]

h) Installed and valid, otherwise 7

DATA ENCRYPTION KEY

FCBCRYPTO software provides 5AES(128, 192, 256)[10] encryption technique for char, nchar, varchar2, nvarchar2, string, blob, clob, nclob, raw, long, long raw, bfile data types and modified OTP[11] encryption technique for number, float, date, timestamp data types. Any binary number data types aren't supported. AES(128, 192, 256) means a 16, 24 or 325 symbol base cryptographic key must be used. FCBCRYPTO software uses the base cryptographic key as a common key as for AES so for OTP technique. FCBCRYPTO software accepts 3 ways to generate the base cryptographic key:

```
 [12]Step 1.
 …
 a. generation by hands (without whitespaces please)
 b. auto-generation by fcbcrypto/bin/../bin/pwgen.sh[13] utility
 c. auto-generation by dd if=/dev/urandom bs=16(24,32) count=1 2>/dev/null |
openssl[14] base64 | sed s/[=O/\]//g | cut -b1-16(24,32) | tee
 d. skip any generation and use existed file ((choose this option if you want to
preserve existed key file)
```

An example of successful base cryptographic key generation:

```
[15]>>>>>> Keyfile is -rw-r--r--. 1 oracle dba 17 Feb 2 12:35
/ora01/app/oracle/admin/orcl/dpdump/key.txt
>>>>>> Key is V4BsPz2zW9HVMC3P
```

The base cryptographic key is a minimum you need to have to perform data encryption/decryption.

Because of before encryption data get conversion (excepting bfile, number, float, date and timestamp types) to AL32UTF8 format, FCBCRYPTO encrypted data can be migrated (for example via Oracle impdp/expdp utility) to the database with a different character set without any symbol corruptions.

8 DBMS_CRYPTO
9 UTL_COMPRESS
10 Advanced Encryption Standard
11 One-time pad
12 an example from installation process
13 Pronounceable password generator. fcbcrypto/bin/pwgen.sh script calls pre-build OS dependent pwgen utility.
14 General-purpose cryptography library
15 an example from installation process

CORE CODE SIGNING

Core code signing is a FCBCRYPTO software feature starting from v. 1.2.379. The reason of that feature's presence is preventing FCBCRYPTO software unauthorized execution. How is it made? When you download FCBCRYPTO software from web site, the main part of software, i.e. **PKG_FCBCRYPTO** package, is fully valid, but dysfunctional. This is because of a very small non-executable encrypted part of **PKG_FCBCRYPTO** package is located at the **TBL_FCBCRYPTO_CE** table in **B_CORE** blob field. That small non-executable encrypted part is the unsigned core code. Without the executable, i.e. signed, core code, **PKG_FCBCRYPTO** package, staying valid from the PL/SQL language point of thew view, can't perform any operations. To make unsigned core code signed and executable, when you perform initial FCBCRYPTO software's installation, you're asked for some steps (in real installation between these steps can be placed other steps):

a.

```
CORE CODE SIGNING:

Hereafter /.../key_ce.txt file will be used for CORE CODE signing. Do you want to
create and fill it on your own or auto generate it (file will be created and auto-
generated key will be inserted into it)?

  a. generation by hands (without whitespaces please)
  b. auto-generation by /home/oracle/crypt/aes/bin/../bin/pwgen.sh utility
  c. auto-generation by dd if=/dev/urandom bs=32 count=1 2>/dev/null | openssl
base64 | sed s/[=O/\]//g | cut -b1-32 | tee
 d. skip any generation and use existed file (choose this option if CODE SIGNING
was once done)
```

As a result of any of your choices from above you must have key_ce.txt key file (see **l_KEYFILE_CE** variable from .dbvariables file) in **l_KEYFILE_DIRECTORY**[4]. key_ce.txt key contains a key for the core code signing

b.

```
Continue to create primary objects? (y/n)y

If it's a fresh installation please answer 'yes' to re-|create tbl_FCBCRYPTO_CE
table. If it's not a fresh installation and you already have tbl_FCBCRYPTO_CE
table with the signed core code please answer 'no'. See please more details at
/home/oracle/crypt/aes/sql/../doc/FCBCryptoUNIXInstallationAndUserGuide.pdf file.
Thus 'yes' means primary objects and tbl_FCBCRYPTO_CE table will be re-|created.
'no' means primary objects will be re-|created, but tbl_FCBCRYPTO_CE table will
remain as is.

Continue? (y/n)
```

c.

```
CORE CODE SIGNING:

Next step is about CORE CODE signing by the ............... key.
The key is stored in /.../.../key_ce.txt file. If you are just making software
installation yet one time and you did signing before please skip this step.
Otherwise please answer 'yes' or pkg_FCBCRYPTO package will remain valid but
dysfunctional.

Do CORE CODE signing? (y/n)
```

Here unsigned core code, stored in **TBL_FCBCRYPTO_CE.B_CORE,** gets a sign, i.e. a key from the key_ce.txt file. Also your database gets a “watermark”. The “watermark” is a hidden sign inside the database. The “watermark” lets detect that core code signing was once done. The “watermark” doesn’t impact any database features or performance. In case of success and first software installation for you see a message

```
>>>>>> PL/SQL procedure successfully completed.

>>>>>> Now core code of FCBCRYPTO sofware is signed by the key from the
>>>>>> /.../key_ce.txt key file. This means execution of
>>>>>> FCBCRYPTO sofware is locked on the key and the current database.
>>>>>> In case of the key is lost or tbl_FCBCRYPTO_CE table is truncated or
>>>>>> corrupted pkg_FCBCRYPTO package remains valid but dysfunctional.
>>>>>> Because of that, and it's
>>>>>> very important, please make a backup copy of the
>>>>>>      /.../key_ce.txt
>>>>>> key file and the
>>>>>>      tbl_FCBCRYPTO_CE
>>>>>> table now. See please more details at
>>>>>> /home/oracle/crypt/aes/sql/../doc/FCBCryptoUNIXInstallationAndUserGuide.pdf
file.
```

In case more then the one try to make core code signing you see a message

```
Prev. core code signing occasion detected. Re-signing is impossible and canceled.
```

Core code signing summary

- core code signing is not about data encryption/decryption
- core code signing prevents unauthorized execution of any **PKG_FCBCRYPTO** package functions or procedures relating to data encryption/decryption
- core code signing must be performed during the first FCBCRYPTO software installation only
- core code signing key is located in key_ce.txt key file (see **l_KEYFILE_CE** variable from .dbvariables file)
- core code signing key is absolutely different to the base and additional cryptographic keys
- **TBL_FCBCRYPTO_CE.B_CORE** field contains not executable, i.e. unsigned, encrypted PL/SQL core code when the installation starts first
- **PKG_FCBCRYPTO** package is valid, but dysfunctional without executable, i.e. signed, core code from **TBL_FCBCRYPTO_CE.B_CORE**
- core code, i.e. **TBL_FCBCRYPTO_CE.B_CORE** data, get signed, i.e. executed, during successful fist software installation
- in case of successful fist software installation your database gets a “watermark”
- “watermark” is a hidden sign inside database
- “watermark” doesn’t affect database performance or any database features

- “watermark” prevents any try to make yet one core code signing
- thus second core signing is impossible by available and legal ways via your copy of FCBCRYPTO software
- please make backup copies of the core code signing key and the signed **TBL_FCBCRYPTO_CE.B_CORE** data right after successful FCBCRYPTO software installation
- in case of signed **TBL_FCBCRYPTO_CE.B_CORE** data is lost, but the core code signing key is not you can get in contact with the FCBCRYPTO software owner, send the the core code signing key and get a new signed-by-your-key **TBL_FCBCRYPTO_CE.B_CORE** data
- in case of the core code signing key is lost, but signed **TBL_FCBCRYPTO_CE.B_CORE** data is not you can get in contact with the FCBCRYPTO software owner, send the signed **TBL_FCBCRYPTO_CE.B_CORE** data and get a new core code signing key

How core code signing works

- you perform initialization procedure from **KEY MANAGEMENT** chapter.
- initialization procedure reads a key from the key_ce.txt key file, encrypts the key and places it into in-memory context
- initialization procedure reads signed encrypted PL/SQL code from **TBL_FCBCRYPTO_CE.B_CORE** and places it into in-memory context
- initialization procedure detects a database “watermark” presence
- if database “watermark” is present, a key from key file and a key from **TBL_FCBCRYPTO_CE.B_CORE** are equal then authorization passed successfully and **PKG_FCBCRYPTO** package functions or procedures can encrypt or decrypt data.
- if the condition above is not complied, **PKG_FCBCRYPTO** package functions or procedures can’t encrypt or decrypt data.

BFILE ENCRYPTION

BFILE stands for LOB data objects stored in operating system files, outside the database tablespaces. FCBCRYPTO software provides a way to encrypt and decrypt BFILE’s. For that OpenSSL toolkit is used. A quick **OpenSSL** toolkit presence verifying can be done as

```
$ openssl version
OpenSSL 1.0.1e-fips 11 Feb 2013
```

Also **OpenSSL** usage means a need to have a “bridge” module to connect an **OpenSSL** utility and an Oracle database. FCBCRYPTO software provides two kind of the “bridge” modules: a module written in C language and a module written in JAVA language. During FCBCRYPTO software installation you can choose any of them. The “bridge” modules are provided as the source code files. That means you can examine source files, edit them and compile them in case of your needs.

If you choose C language “bridge” module then module will be built from scratch by C compiler into a shared library and copied to the **I_EXT_LIB_DIRECTORY** (see .dbvariables file)[16]. Also **EXTPROC_DLLS=ANY** entry will be inserted into **$ORACLE_HOME/hs/admin/extproc.ora** file.

If you choose JAVA language “bridge” module two JAVA database objects will be created inside the database.

16 see src/extlilb/fcbcrypto.sh file how it’s made

Please look at/inside the files for more details:

- fcbcrypto/sql/cre_jsr_fcbcrypto.sql
- fcbcrypto/sql/cre_jsr_fcbcrypto_feedback.sql
- fcbcrypto/sql/cre_lib_fcbcrypto.sql
- fcbcrypto/src/fcbcrypto.c

Thus hardware, operating system and database prerequisites to successfully use BFILE encryption are:

a) UNIX like operating systems, i.e. IBM AIX, SUN/Oracle Solaris, Linux, HP HP-UX[17]
b) installed and valid **OpenSSL** toolkit
c) **openssl** utility is available via **$PATH** environment variable
d) C language compiler to build C language “bridge” module or installed and valid Oracle Database Java subsystem to use JAVA language “bridge” module

```
 [18]Step 4.

Now there is a need to choose a setup connectivity between Oracle database and
/usr/bin/openssl OpenSSL toolkit binary. Please choose one of below:

      a. Oracle database heterogeneous service i.e. entry SET EXTPROC_DLLS=ANY
will be inserted into
/ora01/app/oracle/product/12.1.0/server_se_1/hs/admin/extproc.ora file and
fcbcrypto/bin/../src/extlib/fcbcrypto.so library will be built from scratch and
copied to /tmp catalog
      b. Recommended. Oracle database Java subsystem i.e Java database source
object will be created
      c. Do nothing and skip this setup
```

If a C language “bridge” module was selected[19]

```
>>>>>> Oracle database heterogeneous service connectivity has been selected.

Please make sure Oracle database binaries' owner has read,write permission on
/tmp[20] directory and C compiler is available

Continue? (y/n)y

>>>>>> Building /tmp/fcbcrypto.so shared library from scratch via
fcbcrypto/bin/../src/extlib/fcbcrypto.sh call ...

>>>>>> gcc (GCC) 4.4.7 20120313 (Red Hat 4.4.7-18) GNU C/C++ compiler for Linux is
running...

>>>>>> -rwxr-xr-x. 1 ***** dba 6819 Feb 5 13:36 /tmp/fcbcrypto.so
>>>>>> /tmp/fcbcrypto.so: ELF 64-bit LSB shared object, x86-64, version 1 (SYSV),
dynamically linked, not stripped
```

17 BFILE encryption wasn’t tested on HP HP-UX but it should work
18 an example from installation process
19 Linux example
20 /tmp value comes from **l_EXT_LIB_DIRECTORY** variable from .dbvariables file

```
>>>>>> Hereafter /tmp/fcbcrypto.so shared library will be used
```

If a JAVA language “bridge” module was selected and due to BFILE encryption is based on the external operating system utilities’ calls there is a need to grant external operating system utilities’ calls to the FCBCRYPTO software owner (see **I_PACKAGE_OWNER** variable from .dbvariables file), i.e.

```
[21]>>>>>> Please don't forget to execute under sys account:

SQL> call dbms_java.grant_permission( upper('FCBCRYPTO software owner'),
'SYS:java.io.FilePermission', '/usr/bin/openssl', 'execute');
SQL> call dbms_java.grant_permission( upper('FCBCRYPTO software owner'),
'SYS:java.io.FilePermission', '/bin/rm', 'execute' );
SQL> call dbms_java.grant_permission( upper('FCBCRYPTO software owner'),
'SYS:java.io.FilePermission', '/bin/mv', 'execute' );
SQL> call dbms_java.grant_permission( upper('FCBCRYPTO software owner'),
'SYS:java.io.FilePermission', '/bin/cat', 'execute' );
SQL> call dbms_java.grant_permission( upper('FCBCRYPTO software owner'),
'SYS:java.io.FilePermission', '/usr/bin/wc', 'execute');
SQL> call dbms_java.grant_permission( upper('FCBCRYPTO software owner'),
'SYS:java.io.FilePermission', '/usr/bin/diff', 'execute');
```

If your database is 12c or 11g and 19721304 patch was applied please make sure JAVA DEVELOPMENT feature (if it is) is allowed

```
SQL> select upper(JAVA_DEV_ALLOWED) from sys.java_dev_status;

UPP
---
YES
```

If it’s not allowed please allow it

```
SQL> exec dbms_java_dev.enable;
```

Without allowed JAVA DEVELOPMENT feature (if it is) JAVA language “bridge” module doesn’t work on 12c at least.

There are four BFILE functions:

- fnc_encbfile_bfibfo_disk (p_bfile in bfile) return bfile
- fnc_decbfile_bfibfo_disk (p_bfile in bfile) return bfile
- fnc_decbfile_bfibo (p_bfile in bfile) return blob
- fnc_is_bfile_encrypted_disk (p_bfile in bfile) return simple_integer

BFILE to BFILE encryption via ***PKG_FCBCRYPTO.FNC_ENCBFILE_BFIBFO_DISK*** *function*

If

- **openssl** utility is present
- BFILE encryption/decryption is activated[22]

21 an example from installation process
22 see g_bfile row in **TBL_FCBCRYPTO_SETTING** table from **SETTINGS** chapter

- BFILE is not null
- BFILE was not encrypted by FCBCRYPTO software, i.e. first 8 symbols are not equal to **Salted__**

then steps below are performed when **PKG_FCBCRYPTO.FNC_ENCBFILE_BFIBFO_DISK** function is called:

- insert processing info into **TBL_FCBCRYPTO_BFILE** table if

```
   SQL> select count(*)
          from TBL_FCBCRYPTO_SETTING
         where S_NAME= 'g_bfile_process_tracking' and S_ACTIVE= 'Y';
```

  is equal to 1
- remove BFILE.be[23].sha1 file
- remove BFILE.ae[24].sha1 file
- calculate BFILE check sum by **openssl** utility and write out it to BFILE.be.sha1[25] file
- BFILE file encryption by **openssl** utility into BFILE.session_id temporary file
- remove BFILE file
- move BFILE.session_id file to BFILE file
- calculate BFILE file check sum by **openssl** utility and write out it to BFILE.ae.sha1
- insert processing info into **TBL_FCBCRYPTO_BFILE** table if

```
   SQL> select count(*)
          from TBL_FCBCRYPTO_SETTING
    where S_NAME= 'g_bfile_process_tracking' and S_ACTIVE= 'Y';
```

  is equal to 1

After encryption BFILE related files look like

```
-rw-r--r--. 1 oracle dba      93 Feb  8 09:40 BFILE.be.sha1
-rw-r--r--. 1 oracle dba      93 Feb  8 09:40 BFILE.ae.sha1
-rw-r--r--. 1 oracle dba 1075584 Feb  8 09:40 BFILE
```

*BFILE to BFILE decryption via **PKG_FCBCRYPTO.FNC_DECBFILE_BFIBFO_DISK** function*

If

- **openssl** utility is present
- BFILE encryption/decryption is activated[26]
- BFILE is not null
- BFILE was encrypted by FCBCRYPTO software, i.e. first 8 symbols are equal to **Salted__**

then steps below are performed when **PKG_FCBCRYPTO.FNC_DECBFILE_BFIBFO_DISK** function is called:

- Insert processing info into **TBL_FCBCRYPTO_BFILE** table if

```
   SQL> select count(*)
          from TBL_FCBCRYPTO_SETTING
         where S_NAME= 'g_bfile_process_tracking' and S_ACTIVE= 'Y';
```

23 “be” stands for “before encryption”
24 “ae” stands for “after encryption”
25 sha1 relates to g_digest row in **TBL_FCBCRYPTO_SETTING** from **SETTINGS** chapter
26 see g_bfile row in **TBL_FCBCRYPTO_SETTING** table from **SETTINGS** chapter

is equal to 1

- remove BFILE.bd[27].sha1 file
- remove BFILE.ad[28].sha1 file
- calculate BFILE check sum by **openssl** utility and write out it to BFILE.bd.sha1[29] file
- BFILE file decryption by **openssl** utility into BFILE.session_id temporary file
- remove BFILE file
- move BFILE.session_id file to BFILE file
- calculate BFILE file check sum by **openssl** utility and write out it to BFILE.ad.sha1
- if check sums before encryption and after decryption is equal, i.e. BFILE.be.sha1 file is equal to BFILE.ad.sha1 file then processing info is inserted into **TBL_FCBCRYPTO_BFILE** table if

```
SQL> select count(*)
       from TBL_FCBCRYPTO_SETTING
      where S_NAME= 'g_bfile_process_tracking' and S_ACTIVE= 'Y';
```

is equal to 1

- if check sums before encryption and after decryption is not equal, i.e. BFILE.be.sha1 file is not equal to BFILE.ad.sha1 file an exception rises

```
ORA-20001 Something goes wrong! Check sums are different before encryption and after decryption...
```

After decryption BFILE related files look like

```
-rw-r--r--. 1 oracle dba      93 Feb  8 09:40 BFILE.be.sha1
-rw-r--r--. 1 oracle dba      93 Feb  8 09:40 BFILE.ae.sha1
-rw-r--r--. 1 oracle dba 1075562 Feb  8 09:40 BFILE
-rw-r--r--. 1 oracle dba      93 Feb  8 09:40 BFILE.bd.sha1
-rw-r--r--. 1 oracle dba      93 Feb  8 09:40 BFILE.ad.sha1
```

*BFILE to BLOB decryption via **PKG_FCBCRYPTO.FNC_DECBFILE_BFIBO** function*

- BFILE to BFILE decryption via **PKG_FCBCRYPTO.FNC_DECBFILE_BFIBFO_DISK** function

- if BFILE returned value is not null, i.e. BFILE was encrypted before and it has been decrypted now
- open BFILE
- load BFILE into BLOB
- return BLOB
- BFILE to BFILE encryption via **PKG_FCBCRYPTO.FNC_ENCBFILE_BFIBFO_DISK** function

- if BFILE returned value is null, i.e. BFILE wasn't encrypted
- open BFILE
- load BFILE into BLOB
- return BLOB

*BFILE is encrypted or it's not via **PKG_FCBCRYPTO.FNC_IS_BFILE_ENCRYPTED_DISK** function*

Function returns "0" if BFILE is not encrypted and "1" if BFILE is encrypted. BFILE is encrypted if first 8 symbols are equal to **Salted__**

27 "bd" stands for "before decryption"
28 "ad" stands for "after decryption"
29 sha1 relates to g_digest row in **TBL_FCBCRYPTO_SETTING** from **SETTINGS** chapter

SECURITY TOKEN ENCRYPTION

Security token (in FCBCRYPTO software context) is any physically attached to USB port inexpensive device on the host where Oracle database is placed. That can be a memory stick, card reader, web camera, external hard drive, mouse or keyboard, even a mobile phone[30]. Security Token Encryption (in FCBCRYPTO software context) means usage unique security token hardware information as an additional cryptographic key to improve cryptographic strength. STE can not be used stand alone. STE is used only with a base cryptographic key together.

How it works:

- Base cryptographic key places at the key file (see g_keyfile in **TBL_FCBCRYPTO_SETTING** table) and at the key file directory (see g_keyfile_dir in **TBL_FCBCRYPTO_SETTING** table)
- Security token is attached to the database host
- When initialization process runs (see **KEY MANAGEMENT** chapter for more details), FCBCRYPTO software

  - reads base cryptographic key from the key file 13
  - reads security token hardware information
  - generates additional security token based cryptographic key
  - mixes two keys and gets the real cryptographic key

- Real cryptographic key is used to encrypt/decrypt data

STE usage advantages

- stolen data and stolen key file[31] with the base cryptographic key don't mean encrypted data can be decrypted
- stolen data and stolen security token don't mean encrypted data can be decrypted, especially when key file is not used
- every modern security token has unique combination of hardware information
- security token can not be duplicated
- USB flash drive file system information doesn't matter if USB flash drive plays security token role

STE usage disadvantages

- lost security token means impossibility to decrypt encrypted data
- Linux operating system only
- **lsusb** utility must be installed, valid and available via **$PATH** variable
- usage of security tokens with the simple serial numbers should be avoided

Also STE usage means a presence of the installed and valid Oracle Database Java subsystem because **lsusb** utility's call is performed via JAVA "bridge" module only. Thus hardware, operating system and database prerequisites to successfully use STE are:

a) Linux operating system
b) Oracle Database Java subsystem is installed and valid
c) presence of **lsusb** utility with a **4755** file permission

30 mouse and keyboard aren't recommended because of lack unique hardware information in most models
31 key file usage can be omitted to harden security. See **KEY MANAGEMENT** chapter how to do that

d) **lsusb** utility is available via **$PATH** environment variable
e) USB device can be detected by **lsusb** utility
f) USB device has a vendor id, a product id and an acceptable serial number information[32]
g) FCBCRYPTO database user is granted to run **lsusb** utility

```
$ ls -la `which lsusb`
-rwsr-xr-x. 1 root root 106K May 11  2016 /usr/bin/lsusb*
```

```
$ stat -c "%a" `which lsusb`
4755
```

```
$ lsusb
Bus 002 Device 016: ID 054c:01bd Sony Corp. MRW62E Multi-Card Reader/Writer
Bus 004 Device 002: ID 046d:c002 Logitech, Inc. M-BA47 [MouseMan Plus]
Bus 004 Device 003: ID 046d:c31c Logitech, Inc. Keyboard K120
```

```
$ lsusb -d 054c:01bd -v
  ...
  idVendor           0x054c Sony Corp.
  idProduct          0x01bd MRW62E Multi-Card Reader/Writer
  ...
  iSerial                 3 0000000B736F <------- good serial number
```

```
$ lsusb -d 046d:c002 -v
  ...
  idVendor           0x046d Logitech, Inc.
  iProduct                2 USB-PS/2 Mouse M-BA47
  ...
  iSerial                 0              <------- no serial number at all
```

```
$ lsusb -d 0bb4:2008 -v
...
  idVendor           0x0bb4 HTC (High Tech Computer Corp.)
  idProduct          0x2008 Android Phone via MTP [Wiko Cink Peax 2]
  ...
  iSerial                 4 0123456789ABCDEF <--- worst serial number
```

```
$ lsusb -d 0930:6544 -v
...
  idVendor           0x0930 Toshiba Corp.
  idProduct          0x6544 TransMemory-Mini / Kingston DataTraveler
  ...
  iSerial                 4 50E549C6931BC171E906AE9B <--- best serial number
```

An example of successful STE setup:

```
[33]Re-detecting acceptable USB attached devices...
```

32 vendor id and product id aren’t used as a part of additional cryptographic key. Serial number is used but that doesn’t mean additional cryptographic key consists of the serial number only.
33 an example from installation process

```
Hardware ID(HID) : Serial Number(SN) : USB Device Name

 054c:01bd : 0000000B736F : Sony Corp. MRW62E Multi-Card Reader/Writer

Please enter Hardware ID(HID) you want to use: 054c:01bd

>>>>>> Correct HID entered. SN is present.
>>>>>> STE feature enabled.
>>>>>> Please don't forget to execute under sys account:

SQL> call dbms_java.grant_permission(upper('FCBCRYPTO software owner'),
'SYS:java.io.FilePermission', '/usr/bin/lsusb', 'execute');
```

SETTINGS

FCBCRYPTO software has a lot of settings. All of them get values during FCBCRYPTO software installation process, but after process' finish you can adjust most of them. All settings are placed at **TBL_FCBCRYPTO_SETTING** table. Let's see:

S_ACTIVE	S_NAME	S_VALUE	S_VALUE_2	S_DESCRIPTION
Y	g_encryption_type	4358	16[5]	SYS.DBMS_CRYPTO.ENCRYPT_AES128 + SYS.DBMS_CRYPTO.CHAIN_CBC + SYS.DBMS_CRYPTO.PAD_PKCS5
N	g_encryption_type	4359	24[5]	SYS.DBMS_CRYPTO.ENCRYPT_AES192 + SYS.DBMS_CRYPTO.CHAIN_CBC + SYS.DBMS_CRYPTO.PAD_PKCS5
N	g_encryption_type	4360	32[5]	SYS.DBMS_CRYPTO.ENCRYPT_AES256 + SYS.DBMS_CRYPTO.CHAIN_CBC + SYS.DBMS_CRYPTO.PAD_PKCS5
Y	g_keyfile_dir	DATA_PUMP_DIR[4]		directory (see ALL_DIRECTORIE

S_ACTIVE	S_NAME	S_VALUE	S_VALUE_2	S_DESCRIPTION
				S view) where keyfile is placed
Y	g_keyfile	key.txt		keyfile name
Y	g_keyfile_ce	key_ce.txt		core code encryption keyfile name
Y	g_mv	/bin/mv		utility to move (rename) files
Y	g_rm	/bin/rm		utility to remove files or directories
Y	g_diff	/usr/bin/diff		utility to compare files line by line
Y	g_wc	/usr/bin/wc		utility to print newline, word, and byte counts for each file
Y	g_cat	/bin/cat		utility to concatenate files and print on the standard output
Y	g_digest	sha1		OpenSSL toolkit function to generate and verify digital signatures
Y	g_bytes_per_symbol	2		bytes per symbol for clob operation
Y	g_os	Linux		local operating system name
Y	g_context	global		global (SGA based) or local (PGA based) context access
Y	g_str_deflation_ratio	6		0 deflation is off, 1-fast ...9-best is on for varchar2 and nvarchar2 data type
Y	g_raw_deflation_ratio	6		0 deflation is off, 1-fast ...9-best is on for raw data type

S_ACTIVE	S_NAME	S_VALUE	S_VALUE_2	S_DESCRIPTION
Y	g_blob_deflation_ratio	6		0 deflation is off, 1-fast ...9-best is on for blob data type
Y	g_clob_deflation_ratio	6		0 deflation is off, 1-fast ...9-best is on for clob data type
Y	g_service_function_user_list	SYSTEM		user list who authorized are to run service routines: prc_init, prc_deinit, fnc_usb_ste_present
Y	g_bfile_process_tracking	g_bfile_process_tracking		in case of using please make sure package owner has select, insert grants on TBL_FCBCRYPTO_BFILE table
Y	g_bfile	/usr/bin/openssl	128	BFILE encryption via utility from OpenSSL cryptography and SSL/TLS toolkit https://www.openssl.org/
Y	g_openssl_connection	internal_java_source[19]	jsr_fcbcrypto	database object name
Y	g_lsusb	/usr/bin/lsusb		utility for displaying information about USB buses in the system and the devices connected to them
Y	g_ste	0B72F364FDE4766		STE feature
Y	g_encrypted_context_base_key			Base key is stored in context as encrypted
Y	g_encrypted_context_add_key			Additional key is stored in context as encrypted

g_encryption_type is a kind of AES encryption to use. De-/activate any of them via “Y” or “N” value in the S_ACTIVE field. Only one “Y” value must be set. S_VALUE_2 is a base cryptographic key size5. Please change neither S_VALUE nor S_VALUE_2 by hands.

g_keyfile_dir is a **l_KEYFILE_DIRECTORY**4 value from .dbvariables file. S_ACTIVE value is always “Y”.

g_keyfile is a **l_KEYFILE** value from .dbvariables file. S_ACTIVE value is always “Y”.

g_keyfile_ce is a **l_KEYFILE_CE** value from .dbvariables file. S_ACTIVE value is always “Y”.

g_mv, g_rm, g_diff, g_wc, g_cat, g_lsusb are paths to the correspondent utilities. Paths come from .osvariables file. S_ACTIVE value is always “Y”.

g_digest is an **OpenSSL** toolkit function to generate and verify digital signatures of encrypted/decrypted BFILE’s. Default value is sha1[34]. Change it if you need. S_ACTIVE value is always “Y”.

g_bytes_per_symbol is a deprecated variable

g_os is an auto-detected operating system common name. Please change in case of migration to the other OS only. S_ACTIVE value is always “Y”.

g_context can have global (SGA based) or local, i.e. session only (PGA based) context access[35] value. Default is global. Change it if you need. S_ACTIVE value is always “Y”.

g_str_deflation_ratio, g_raw_deflation_ratio, g_blob_deflation_ratio, g_clob_deflation_ratio is compression quality (for the correspondent data types) in the range from 0 to 9, i.e. 0 = no compression, 1 = fast compression, 9 = best compression. Default value is 6. Change it if you need. S_ACTIVE value is always “Y”. See **DATA COMPRESSION** chapter.

g_service_function_user_list 22 lists users who authorized are to run key management service routines: **PKG_FCBCRYPTO**.**PRC_INIT** procedure, **PKG_FCBCRYPTO**.**PRC_DEINIT** procedure, **PKG_FCBCRYPTO**.**FNC_USB_STE_PRESENT** function (see more details in **KEY MANAGEMENT** chapter). Default value is a FCBCRYPTO software owner (see **l_PACKAGE_OWNER** from .dbvariables file). You can add anyone or remove all of them. S_ACTIVE value is always “Y”.

g_bfile[36]. When S_ACTIVE value is “Y” then S_VALUE is a path to **openssl** utility (if it’s present) and BFILE encryption (see **BFILE ENCRYPTION** chapter) is active. When S_ACTIVE value “N” then S_VALUE is still the path to the **openssl** (if it is present) utility but BFILE encryption is not active.

g_openssl_connection. It it’s present then S_VALUE shows C language “bridge” module or JAVA language “bridge” module will be used for BFILE encryption (see **BFILE ENCRYPTION** chapter). S_VALUE_2 shows database “bridge” module object name. S_ACTIVE is always “Y”. If g_openssl_connection string is not present this means BFILE encryption is not active.

g_bfile_process_tracking. If BFILE encryption is active bfile encryption steps, i.e. “start”, “processing”, “finish” will be logged at **TBL_FCBCRYPTO_BFILE** table. S_ACTIVE value can be “Y” or “N”. Change it if you need. In case of “Y” please make sure FCBCRYPTO software owner has select, insert grants on T**BL_FCBCRYPTO_BFILE** table.

34 a wikipedia article
35 context creation
36 g_bfile S_ACTIVE value has highest priority in order to get BFILE encryption is active or it is not

g_ste[37] When g_ste row is present it contains: a security token hash sum (see **SECURITY TOKEN ENCRYPTION** chapter) and S_ACTIVE value is "Y". This means security token encryption was setup and it is used. Setup S_ACTIVE value to "N" in order not to use security token encryption. When g_ste row is not present this means security token encryption wasn't setup and it is not used. Note: A value you see in S_VALUE field in the table above is a fake value.

g_encrypted_context_base_key defines to perform (S_ACTIVE = "Y") or not to perform (S_ACTIVE = "N") base cryptographic key context encryption, i.e. if base cryptographic key is Z3Wx!2#@%^&(CCZF in case of S_ACTIVE = "N" it will be stored as Z3Wx!2#@%^&(CCZF value in context, in case of S_ACTIVE = "Y" it will be stored as 3E40C84792503556AEC4A88E7AFE383E2CDAB00BDE344C457DA52DDD96F3E203911E2915499CC20C7D54CBA79A985FCE value.

g_encrypted_context_add_key defines to perform (S_ACTIVE = "Y") or not to perform (S_ACTIVE = "N") additional cryptographic key context encryption.

AFTER INSTALLATION

When installation finishes you have following objects in FCBCRYPTO software schema

- **PKG_ FCBCRYPTO** package[38]
- **TBL_FCBCRYPTO_CE** table[39]
- **FNC_ FCBCRYPTO_HASHTYPE**[40] function
- **LIB_ FCBCRYPTO** external library[41]
- 19**JSR_FCBCRYPTO** java source[42]
- **JSR_FCBCRYPTO_FEEDBACK** java source[43]
- **TBL_FCBCRYPTO_SETTING** table[44]
- **TBL_FCBCRYPTO_BFILE** table[45]

PKG_ FCBCRYPTO package is a main part of FCBCRYPTO software. **PKG_ FCBCRYPTO** package consists of two parts: a package specification and a wrapped package body.

Correspondent files:

- fcbcrypto/sql/cre_pkg_fcbcrypto.pks
- fcbcrypto/sql/cre_pkg_fcbcrypto.pkb

PKG_ FCBCRYPTO package provides following encryption, decryption and service routines:

CHAR, VARCHAR2, STRING

- function fnc_encvchr (p_data in varchar2, l_custom_deflation in pls_integer default -1) return varchar2

37 g_ste S_ACTIVE value has highest priority in order to get security token encryption is active or it is not
38 **PKG_ FCBCRYPTO** package validity depends on the all objects below excepting **TBL_FCBCRYPTO_BFILE** table
39 see **CORE CODE SIGNING** chapter for more details
40 provides maximum available cryptographic hash algorithm for the current version of Oracle database
41 usage depends on your prev. choice
42 usage depends on your prev. choice
43 usage depends on your prev. choice
44 see **SETTINGS** chapter for more details
45 table is created in dependency on your choice

- function fnc_decvchr (p_data in varchar2, l_custom_deflation in pls_integer default -1) return varchar2

NCHAR, NVARCHAR2

- function fnc_encnvch (p_data in nvarchar2, l_custom_deflation in pls_integer default -1) return nvarchar2
- function fnc_decnvch (p_data in nvarchar2, l_custom_deflation in pls_integer default -1) return nvarchar2

NUMBER, FLOAT (in -/+9.9*10^36 range)

- function fnc_encnum (l_value in number) return number
- function fnc_decnum (l_value in number) return number

DATE

- function fnc_encdate (l_value in date) return date
- function fnc_decdate (l_value in date) return date

TIMESTAMP WITH OR WITHOUT TIMEZONE[46]

- function fnc_enctstp (l_value in timestamp / timestamp with time zone) return timestamp / timestamp with time zone
- function fnc_dectstp (l_value in timestamp / timestamp with time zone) return timestamp / timestamp with time zone

BLOB

- function fnc_encblob (p_blob in blob, l_custom_deflation in pls_integer default -1) return blob
- function fnc_decblob (p_blob in blob, l_custom_deflation in pls_integer default -1) return blob

CLOB

- function fnc_encclob (p_clob in clob, l_custom_deflation in pls_integer default -1) return clob
- function fnc_decclob (p_clob in clob, l_custom_deflation in pls_integer default -1) return clob

NCLOB

- function fnc_encnclob (p_nclob in nclob, l_custom_deflation in pls_integer default -1) return nclob
- function fnc_decnclob (p_nclob in nclob, l_custom_deflation in pls_integer default -1) return nclob

RAW

- function fnc_encraw (p_raw in raw, l_custom_deflation in pls_integer default -1) return raw
- function fnc_decraw (p_raw in raw, l_custom_deflation in pls_integer default -1) return raw

LONG RAW

- function fnc_enclraw (p_lraw in long raw, l_custom_deflation in pls_integer default -1) return long raw
- function fnc_declraw (p_lraw in long raw, l_custom_deflation in pls_integer default -1) return long raw

LONG

46 encryption is limited by milliseconds, i.e. microseconds and less aren’t encrypted

- function fnc_enclong (p_long in long, l_custom_deflation in pls_integer default -1) return long
- function fnc_declong (p_long in long, l_custom_deflation in pls_integer default -1) return long

BFILE is encrypted or it's not. See BFILE ENCRYPTION chapter

- function fnc_is_bfile_encrypted_disk (p_bfile in bfile) return simple_integer

BFILE. See BFILE ENCRYPTION chapter

- function fnc_encbfile_bfibfo_disk (p_bfile in bfile) return bfile
- function fnc_decbfile_bfibfo_disk (p_bfile in bfile) return bfile

BFILE to BLOB. See BFILE ENCRYPTION chapter

- function fnc_decbfile_bfibo (p_bfile in bfile) return blob

Key management routines. See KEY MANAGEMENT chapter

- procedure prc_init (l_in_key in varchar2 default null)
- procedure prc_deinit
- function fnc_usb_ste_present (deinit_if_missed simple_integer default 0, init_if_reattached simple_integer default 0) return simple_integer

Integrity checking routines. See TAMPER-PROOFING chapter

- function fnc_intergrity return varchar2

"**enc**" in routine names stands for encryption. "**dec**" in routine names stands for decryption

l_custom_deflation input parameter means compression ratio in the range from 0=no compression, 1=fastest compression to 9=best compression. Omitting, i.e. using default -1 value, means correspondent g_*_deflation_ratio value from **TBL_FCBCRYPTO_SETTING** table (see **SETTINGS** chapter) will be used.

KEY MANAGEMENT

FCBCRYPTO software provides centralized in-memory key management process. Key management initialization is a mandatory first step to start data encryption/decryption[47]. Cryptographic keys or cryptographic key traces aren't stored or presented in the constant database objects like tables. Cryptographic keys are distributed to the FCBCRYPTO decryption/encryption functions via Oracle database in-memory object only. That object is context (a set of application-defined attributes that validates and secures an application). Encryption/decryption can't be performed without in-memory loaded cryptographic keys. How it works in details.

Initialization

Database user having privileges to execute **PKG_FCBCRYPTO** package runs **PKG_FCBCRYPTO**.**PRC_INIT**

47 encryption/decryption **PKG_FCBCRYPTO** package routines don't work without key management initialization

procedure. **PKG_FCBCRYPTO**.**PRC_INIT** procedure accepts either

- a null input parameter and then procedure starts to read base cryptographic key from the key file (see g_keyfile in **TBL_FCBCRYPTO_SETTING** table) from the key file directory (see g_keyfile_dir in **TBL_FCBCRYPTO_SETTING** table) or
- a 16, 24 or 32 symbol base cryptographic key as an input parameter[48]

```
SQL> exec pkg_fcbcrypto.prc_init;
or
SQL> exec pkg_fcbcrypto.prc_init('cHBmNz7cfTqH5t82VlXvjd8LdL45XccL');
```

If user is not in allowed-to-run-key-magement-routine-user list, i.e.

```
SQL> select S_VALUE as USERS_ALLOWED_TO_RUN_KEY_MANAGEMENT_ROUTINES
       from TBL_FCBCRYPTO_SETTING
      where S_NAME = ‘g_service_function_user_list’ and S_ACTIVE=’Y’
```

then **PKG_FCBCRYPTO**.**PRC_INIT** stops initialization[49] and a message

```
User is not authorized to run prc_init routine.
```

appears.

Otherwise **PKG_FCBCRYPTO** context is re-/created with a global or local access. Context access comes from

```
SQL> select S_VALUE as CONTEXT_ACCESS
       from TBL_FCBCRYPTO_SETTING
      where S_NAME = ‘g_context’ and S_ACTIVE=’Y’
```

After context creation context is filled out by the cryptographic related variables and values. Next step is a security token encryption setup. If security token encryption was activated, i.e.

```
SQL> select (case when count(*)= 1
                  then ‘STE is active’
                  else ‘STE is not active’
             end) as STE_STATUS
       from TBL_FCBCRYPTO_SETTING
      where S_NAME = ‘g_ste’ and S_ACTIVE=’Y’
```

then security token hash sum is calculated. If security token hash sum is equal to this one

```
SQL> select S_VALUE as DEFINED_DURING_INSTALLATION_SECURITY_TOKEN_HASHSUM
       from TBL_FCBCRYPTO_SETTING
      where S_NAME = ‘g_ste’ and S_ACTIVE=’Y’
```

then additional security token based cryptographic key is calculated. Otherwise, i.e. hash sums are different, because attached security token is wrong, nothing appears. If there is no attached security token at all, a

48 This means you don’t need to worry key file (see g_keyfile in **TBL_FCBCRYPTO_SETTING** table) can be stolen
49 Why does that user list exist? Because you may have no intention to provide key management routine executable rights to the user already having executable grant on **PKG_FCBCRYPTO** package.

warning appears

```
initializing STE...

Do not you forget to attach a proper USB device `cause setting table contains
activated STE feature?
```

Context is/is not filled out by the cryptographic related variables and values again in dependence on the previous results.

Next. If **TBL_FCBCRYPTO_BFILE** table is presented then it is truncated.

Here **PKG_FCBCRYPTO**.**PRC_INIT** procedure finishes key management initialization. This means all encryption/decryption **PKG_FCBCRYPTO** package routines (see **AFTER INSTALLATION** chapter) are ready to be used.

An example of successful key management initialization SQL*Plus output by **PKG_FCBCRYPTO**.**PRC_INIT** procedure

```
PKG_FCBCRYPTO context is being re|-created...
[50]signed core code is being initialized...
[51]reading setting table...
[52]initializing STE...
[53]reading ce key file...
reading key file...
context filled up...
bfile table is being truncated...
init done

PL/SQL procedure successfully completed.
```

What if initialization was not performed or performed unsuccessfully and data encryption/decryption functions are called? An exception rises

```
ORA-06502: PL/SQL: numeric or value error
```

Please have a note **FCBCRYPTO software doesn't monitor base cryptographic key presence**, i.e. a presence of the key file (see g_keyfile in T**BL_FCBCRYPTO_SETTING** table) at the key file directory (see g_keyfile_dir in **TBL_FCBCRYPTO_SETTING** table), **in a real-time mode**. Also that means the key file is present or it is not, the key file directory is present or it is not, a security token is present or it is not, FCBCRYPTO software, if it was initialized successfully once, knows nothing about missed key, missed catalog or missed security token, i.e. in-memory context still can contain cryptographic information. Only launched **PKG_FCBCRYPTO**.**PRC_INIT** or **PKG_FCBCRYPTO.FNC_USB_STE_PRESENT** procedures can check missed stuff.

50 see **CORE CODE SIGNING** chapter for details
51 see **SETTINGS** chapter for details
52 see **SECURITY TOKEN ENCRYPTION** chapter for details
53 see **CORE CODE SIGNING** chapter for details

Deinitialization

PKG_FCBCRYPTO.**PRC_DEINIT** procedure simply erases previously filled cryptographic in-memory context information and delete in-memory context. After that any attempt to call **PKG_FCBCRYPTO** package encryption/decryption functions to encrypt/decrypt data gives only an error like

```
ORA-06502: PL/SQL: numeric or value error
```

An example of successful key management deinitialization SQL*Plus output by **PKG_FCBCRYPTO**.**PRC_DEINIT** procedure

```
PKG_FCBCRYPTO context de-initialized

PL/SQL procedure successfully completed.
```

If user is not in a allowed-to-run-key-magement-routine-user list, i.e.

```
SQL> select S_VALUE as USERS_ALLOWED_TO_RUN_KEY_MANAGEMENT_ROUTINES
       from TBL_FCBCRYPTO_SETTING
      where S_NAME = ‘g_service_function_user_list’ and S_ACTIVE=’Y’
```

then **PKG_FCBCRYPTO**.**PRC_DEINIT** procedure cancels deinitialization[54] and a message appears.

```
User is not authorized to run prc_deinit routine.
```

Security token presence verifying

PKG_FCBCRYPTO.FNC_USB_STE_PRESENT function is assigned for that. There are two input parameters: **deinit_if_missed** and **init_if_reattached**. Zero input parameters mean to do nothing.

Security token	**function returns**	**deinit_if_missed = 1**	**init_if_reattached = 1**
missed	0	PKG_FCBCRYPTO.PRC_DEINIT	
present	1		PKG_FCBCRYPTO.PRC_INIT
unknown status	587		

When **PKG_FCBCRYPTO.FNC_USB_STE_PRESENT** function is called and **deinit_if_missed=1** and there is no an attached security token with a hardware hash sum that is equal to the hash sum from **PKG_FCBCRYPTO_SETTING** table, i.e.

```
SQL> select S_VALUE
       from TBL_FCBCRYPTO_SETTING
     where S_NAME = 'g_ste' and S_ACTIVE='Y';

S_VALUE
```

54 Why does that user list exist? Because you may have no intention to provide key management routine executable rights to the user already having executable grant on **PKG_FCBCRYPTO** package.

```
--------------------------------------------------------------------------------
0B72F364FDE4766E9B528A6F07AAA26F0F9FC589A48BE
```

then **PKG_FCBCRYPTO.PRC_DEINIT** is called.

When **PKG_FCBCRYPTO.FNC_USB_STE_PRESENT** function is called, **init_if_reattached=1** and some security token is attached **PKG_FCBCRYPTO.FNC_USB_STE_PRESENT** function calls **lsusb** utility to read security token hardware information and to generate a current security token hash sum. After that function performs comparison of the current security token hardware information hash sum and a hash sum storing in **PKG_FCBCRYPTO_SETTING** table, i.e.

```
SQL> select S_VALUE
      from TBL_FCBCRYPTO_SETTING
     where S_NAME = 'g_ste' and S_ACTIVE='Y';

S_VALUE
--------------------------------------------------------------------------------
0B72F364FDE4766E9B528A6F07AAA26F0F9FC589A48BE
```

If two hash sums are matched **PKG_FCBCRYPTO.PRC_INIT** procedure is called. If they aren’t matched neither warning nor error nor exception appears.

In any case **PKG_FCBCRYPTO.FNC_USB_STE_PRESENT** function returns either “0” when no security token attached or a wrong security stick attached, or “1” when a correct security token attached, or ‘587’.
If user isn’t authorized to run **PKG_FCBCRYPTO.FNC_USB_STE_PRESENT** function, i.e.
user is out of

```
SQL> select S_VALUE as USERS_ALLOWED_TO_RUN_KEY_MANAGEMENT_ROUTINES
       from TBL_FCBCRYPTO_SETTING
      where S_NAME = ‘g_service_function_user_list’ and S_ACTIVE=’Y’
```

a message

```
User is not authorized to run fnc_usb_ste_present routine.
```

appears and “587” returns.

Have a note please. **PKG_FCBCRYPTO.FNC_USB_STE_PRESENT** function does not call **PKG_FCBCRYPTO.PRC_INIT** procedure if function detects **PKG_FCBCRYPTO** context’s presence, i.e. **PKG_FCBCRYPTO.PRC_DEINIT** procedure wasn’t called before by the any way. Thus the most suitable ways to call **PKG_FCBCRYPTO.FNC_USB_STE_PRESENT** function are

```
SQL> select PKG_FCBCRYPTO.FNC_USB_STE_PRESENT(1,1) from dual;
or
SQL> select PKG_FCBCRYPTO.FNC_USB_STE_PRESENT(0,0) from dual;
or
SQL> select PKG_FCBCRYPTO.FNC_USB_STE_PRESENT from dual;
```

Key management summary

- base cryptographic key comes from the key file (see g_keyfile_dir in **TBL_FCBCRYPTO_SETTING**

table) and the key file directory
(see g_keyfile_dir in **TBL_FCBCRYPTO_SETTING** table)
- base cryptographic key can come as an input parameter of **PKG_FCBCRYPTO.PRC_INIT** procedure call also
- additional cryptographic key comes from the security token hardware info
- base cryptographic key is stored in in-memory context after initialization
- additional cryptographic key is stored in the in-memory context
- base and additional cryptographic keys get to the in-memory context via **PKG_FCBCRYPTO.PRC_INIT** procedure call
- base cryptographic key and additional cryptographic key can be guaranteed erased from the memory by **PKG_FCBCRYPTO.PRC_DEINIT** procedure call, the in-memory context's deletion by database administrator or by the database reboot
- real cryptographic key is a derivative of the base and additional cryptographic keys
- real cryptographic doesn't store in the in-memory context where the base and additional cryptographic keys do
- real cryptographic key is calculated every time when data encryption/decryption **PKG_FCBCRYPTO** package routines are called
- data are encrypted and decrypted by the real cryptographic key only
- any keys aren't stored in the constant database objects like tables
- **PKG_FCBCRYPTO.PRC_INIT** procedure must be call only once if context was defined as global[55]
- **PKG_FCBCRYPTO.PRC_INIT** procedure must be call every time when session is created if context was defined as local
- Encryption/decryption is impossible without initial **PKG_FCBCRYPTO.PRC_INIT** procedure call

DATA COMPRESSION

FCBCRYPTO software provides a data compression option for char, nchar, varchar2, nvarchar2, string, blob, clob, nclob, raw, long, long raw SQL and PL/SQL data types. Compression availability is not a goal of FCBCRYPTO software, but a side effect. This effect got a life because of

- impossibility to forecast the size of encrypted data especially for *char* and string data types.
- obligatory conversion for the text contained data to ALT32UTF8 format before encryption

FCBCRYPTO software data compression bases on Oracle **UTL_COMPRESS** package. FCBCRYPTO software user may change compression quality in the range from 0 to 9, i.e. 0 = no compression, 1 = fast compression, 9 = best compression quality. Default value is 6 and it comes from

```
SQL> select S_NAME, S_VALUE
       from TBL_FCBCRYPTO_SETTING
      where S_NAME like 'g_%_deflation_ratio' and S_ACTIVE='Y';

S_NAME                  S_VALUE
----------------------  --------------
g_str_deflation_ratio   6
g_raw_deflation_ratio   6
g_blob_deflation_ratio  6
g_clob_deflation_ratio  6
```

(see **SETTINGS** chapter). Please have a note compression could be ineffective in case of small size data. Also you can use either default ratio or customized ratio via **l_custom_deflation** input parameter of

55 see g_context in **TBL_FCBCRYPTO_SETTING** table

encryption/decryption **PKG_FCBCRYPTO** package routines (see **AFTER INSTALLATION** chapter). Customized ratio has higher priority over default **TBL_FCBCRYPTO_SETTING** table ratio. It's highly not recommended to use **l_custom_deflation** input parameter less than 5 when string field has length less than maximum, i.e. 4000 or 32767 characters, because it can lead to impossibility to store encrypted data.

DEMOS, EXAMPLES AND TESTS

If your answer was "yes"

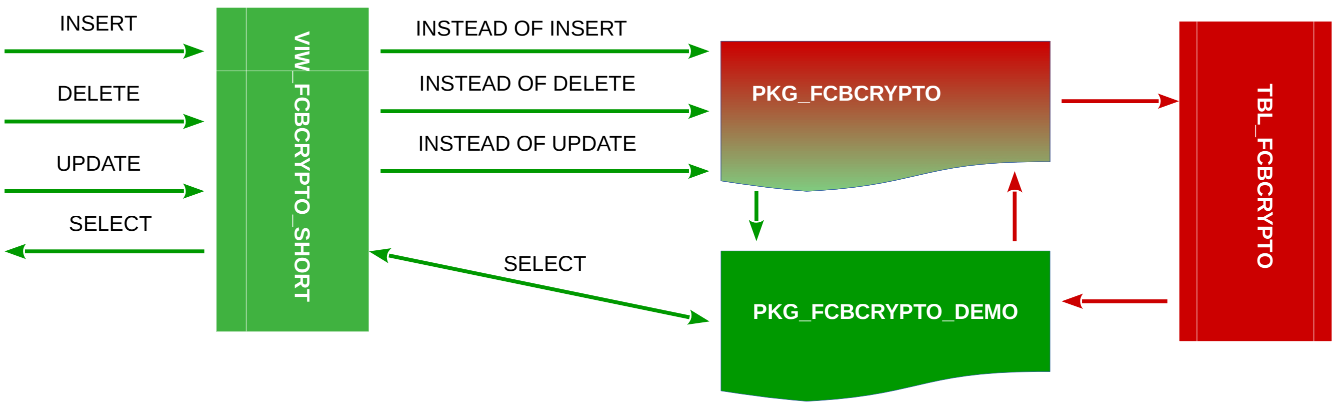
```
>>>>>> Installation of FCBCRYPTO software v.1.2.61 has been finished successfully.

If you choose to continue two files clob.txt and fjmo.pdf will be copied from
fcbcrypto/bin/../dat to "l_KEYFILE_DIRECTORY4" directory in LOB test purposes

Continue to create demo and test objects and run tests? (y/n)
```

you have installed and valid Oracle database demo and test objects (triggers and sequences are not listed):

- **PKG_FCBCRYPTO_DEMO** package
- **VIW_FCBCRYPTO** view
- **VIW_FCBCRYPTO_SHORT** view
- **TBL_FCBCRYPTO** table
- **TBL_FCBCRYPTO_BLOB** table
- **GTT_FCBCRYPTO** global temporary table

So how demos and **PKG_FCBCRYPTO** package in general can be used?

Variant A.



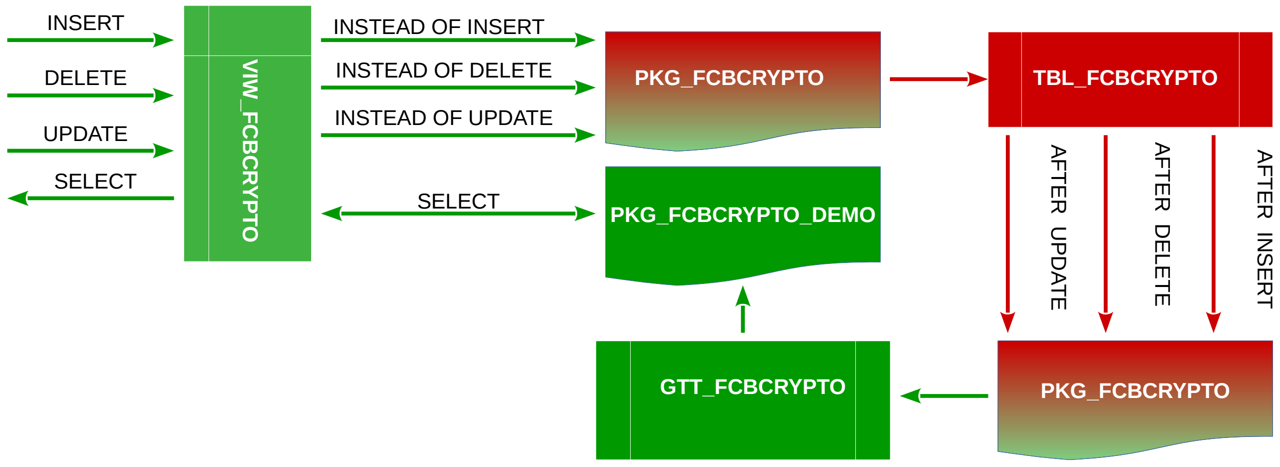
VIW_FCBCRYPTO_SHORT view is based on the statement

```
SQL> select *
```

```
       from table(PKG_FCBCRYPTO_DEMO.FNC_UNVEIL_BULK_SHORT(cursor(select * from
TBL_FCBCRYPTO)));
```

Plain data go to **VIW_FCBCRYPTO_SHORT** view. **VIW_FCBCRYPTO_SHORT** view contains three “instead of insert, update and delete” triggers. When insert, delete or update statements are performed, triggers catch data, encrypt data via **PKG_FCBCRYPTO** package and insert encrypted data into **TBL_FCBCRYPTO** table. **TBL_FCBCRYPTO** table stores encrypted data only. When select statement is performed from **VIW_FCBCRYPTO_SHORT** view a **PKG_FCBCRYPTO_DEMO.FNC_UNVEIL_BULK_SHORT** function requests encrypted data from **TBL_FCBCRYPTO** table, decrypts data via **PKG_FCBCRYPTO** package and return plain data to **VIW_FCBCRYPTO_SHORT** view.

Variant B.



VIW_FCBCRYPTO view is based on the statement

```
SQL> select *
       from table(PKG_FCBCRYPTO_DEMO.FNC_UNVEIL_BULK(cursor(select * from
GTT_FCBCRYPTO)));
```

Plain data go to **VIW_FCBCRYPTO** view. **VIW_FCBCRYPTO** view contains three “instead of insert, update and delete” triggers. When insert, delete or update statements are performed, triggers catch data, encrypt data via **PKG_FCBCRYPTO** package and insert encrypted data into **TBL_FCBCRYPTO** table. **TBL_FCBCRYPTO** table stores encrypted data only. **TBL_FCBCRYPTO** table contains three “after insert, update and delete” triggers also. After data committing triggers call **PKG_FCBCRYPTO** package, **PKG_FCBCRYPTO** package decrypts data and plain decrypted data are inserted into global temporary **GTT_FCBCRYPTO** table. Why global temporary table? Because indexes can be created on it. **GTT_FCBCRYPTO** table contains session level only plain decrypted data. When select statement is performed from **VIW_FCBCRYPTO** view a **PKG_FCBCRYPTO_DEMO.FNC_UNVEIL** function requests data from **GTT_FCBCRYPTO** table and returns plain data to **VIW_FCBCRYPTO** view.

See more details in files

- sql/step_5_test.sql
- sql/cre_pkg_fcbcrypto_demo.pck
- sql/step_7_test.sql
- sql/step_8_test.sql
- sql/step_10_test.sql

- sql/step_11_test.sql
- sql/step_12_test.sql
- sql/step_14_test.sql
- sql/step_15_test.sql
- sql/step_17_test.sql
- sql/step_18_test.sql

VERSION

```
$ sqlplus ******/******
...
SQL> set serveroutput on
SQL> exec pkg_fcbcrypto.prc_about;

FCBCrypto software v.1.2.290
Copyright (c) 2018, Olexandr Siroklyn. All rights reserved.

$ sqlplus ******/******
...
SQL> select pkg_fcbcrypto.fnc_about from dual;

FNC_ABOUT
--------------------------------------------------------------------------------
FCBCrypto software v.1.2.290 Copyright (c) 2018, Olexandr Siroklyn. All rights r
eserved.
```

TAMPER-PROOFING

PKG_FCBCRYPTO package is a tamper-proof featured PL/SQL package. It is protected against reverse engineering and modifications via: wrapping and obfuscation techniques, control sum presence and some other stuffs. Package integrity self-checking can be performed in case of suspicions the package body was unauthorized modified. That modification can be detected via

```
$ sqlplus ******/******
...
SQL> select pkg_fcbcrypto.fnc_integrity from dual;

FNC_INTEGRITY
--------------------------------------------------------------------------------
Integrity check passed. No package code modification detected.

$ sqlplus ******/******
...
SQL> select pkg_fcbcrypto.fnc_integrity from dual;

FNC_INTEGRITY
--------------------------------------------------------------------------------
```

```
Integrity check failed. Package code modification detected.
```

COPYRIGHTS

CONTACTS

Olexandr Siroklyn
Clearwater, FL, USA | +1-727-252-9680