HANSER

Oracle Security in der Praxis

Sicherheit für Ihre Oracle-Datenbank

ISBN 3-446-40436-8

Leseprobe

Weitere Informationen oder Bestellungen unter
http://www.hanser.de/3-446-40436-8 sowie im Buchhandel

4

Datenspeicherung und Datensicherung

4 Datenspeicherung und Datensicherung

In diesem Kapitel geht es darum, wie die Daten auf physikalischer Ebene – also die physikalischen Dateien der Datenbank und ihre Sicherungen – geschützt werden können.

Die Problematik ist ja die: Auf der einen Seite können Sie einen beträchtlichen Aufwand betreiben, um Ihre Daten vor einem unberechtigten Zugriff durch einen Benutzer in der Datenbank abzusichern. Auf der anderen Seite können diese Daten dann beispielsweise mit einem einfachen strings-Kommando von jedem Systemadministrator ausgelesen werden. Deshalb ist es gerade bei sensitiven Daten oft wünschenswert, dass diese Daten auch physikalisch in einer gesicherten Form abgelegt werden. Die hier besprochenen Methoden schützen also vor einem potenziellen Angriff, bei dem der Angreifer physikalischen Zugriff auf die Dateien der Datenbank hat. Ein anderes mögliches Szenario ist der Diebstahl eines Datenbankservers. Zwar sind das Angriffsszenarien, die eher selten vorkommen, aber utopisch sind sie nicht. In der Presse tauchen solche Fälle immer mal wieder auf.

Interessant ist das natürlich auch speziell für Sicherungen, zumal ein Verlust von Sicherungsbändern auf den ersten Blick nicht so tragisch erscheinen mag; die Datenbank läuft ja noch, und das Backup lässt sich wiederholen. Hat der Angreifer aber mal die Sicherung Ihrer Daten, ist das anschließende Recovery der Daten auf dem Rechner des Angreifers nur noch eine einfache Übung. Für manche Firmen wäre so etwas absolut katastrophal.

Schutz gegen diese Szenarien bietet offensichtlich das Verschlüsseln der Daten. Damit wird gewährleistet, dass das Umgehen der Datenbankautorisierung durch Zugriff auf Dateiebene nicht mehr möglich ist. Idealerweise werden sowohl die Dateien der Datenbank als auch die Sicherungen verschlüsselt.

Speziell an dieser Stelle wird oft die Frage gestellt, warum der Zugriff durch den Systemadministrator überhaupt verhindert werden soll. Das Argument hier ist: Wenn Sie kein Vertrauen zu Ihrem Administrator haben, haben Sie kein technisches Problem, sondern ein personelles. Egal, wie Sie es anstellen, letzten Endes kommen Sie immer an einen Punkt, an dem Sie jemandem einfach vertrauen müssen. Wenn Sie Ihrem Systemadministrator nicht trauen können, dann haben Sie wahrscheinlich den falschen Administrator (oder umgekehrt).

Andererseits: Wenn es sich um sehr sensitive Daten handelt, zum Beispiel Krankenakten, ist es absolut einleuchtend, dass diese Daten nur für den Patienten und seinen behandelnden Arzt sichtbar sein sollen.

Aber das ist nicht der einzige denkbare Grund für eine Verschlüsselung der Dateien: Beispielsweise können Richtlinien in Ihrer Firma oder gesetzliche Vorgaben bestehen, die ihrerseits die Verschlüsselung bestimmter Daten zwingend fordern.

Die hier beschriebenen Methoden stellen sicher, dass diese Anforderungen erfüllt und die Richtlinien erfolgreich umgesetzt werden können. Allerdings muss dann, falls Sie dem DBA

wirklich nicht trauen wollen oder können, ein Sicherheitsadministrator eingesetzt werden, der die notwendigen Verschlüsselungs-Keys verwaltet.

Für das Verschlüsseln der Daten in der Datenbank bietet Oracle zwei Möglichkeiten: transparente Datenverschlüsselung und die Verschlüsselung mittels des DBMS_CRYPTO PL/SQL Package.

Für die Verschlüsselung der Sicherungen kann seit Oracle Version 10.2 Oracle RMAN verwendet werden. Die Sicherung kann dann auf verschiedene Arten verschlüsselt werden.

Datenverschlüsselung ist insbesondere für Daten, die als Zeichenketten vorliegen, sinnvoll. Diese Art von Daten ist in den Dateien und Sicherungen direkt sichtbar.

4.1 Verschlüsselung der Daten innerhalb der Datenbank

Eine erste und einfache Sicherung gegen unberechtigten Zugriff auf Betriebssystemebene ist das korrekte Setzen der Dateizugriffsrechte. Das Lesen und Schreiben von Datenbankdateien geschieht niemals direkt durch den Benutzer, sondern immer durch die entsprechenden Datenbankprozesse. Es sind dies der Database-Writer-Prozess (DBW<n>) und Direct-Loader-Prozesse. Das bedeutet, es genügt vollkommen, die Dateien der Datenbank nur für den entsprechenden Benutzer – meistens oracle oder so ähnlich genannt – zu öffnen. Unter Unix hat dieser Benutzer auch noch eine spezielle Gruppe – meistens dba genannt –, die für normale Benutzer gesperrt sein sollte. Unter diesen Voraussetzungen sorgt das Kommando: chmod 600 , wenn es auf Datenbankdateien angewandt wird, effektiv dafür, dass nur noch der Oracle-Benutzer diese Dateien auf Betriebssystemebene lesen kann.

Einen Systemadministrator hindert das natürlich nicht am Lesen der Datenbankdateien, da er sich einfach die notwendigen Rechte geben kann. Aber zumindest der Zugriff durch gewöhnliche Benutzer ist damit effektiv verhindert.

Die Verschlüsselung der Datenbank auf Betriebssystemebene mit entsprechenden Tools wie zum Beispiel PGP oder die in Windows XP angebotene Variante der Dateiverschlüsselung über die erweiterten Dateiattribute kann für die Verschlüsselung von Datenbankdateien nicht verwendet werden; damit können Sie sich die Datenbank recht schnell zerstören. Wohl dem, der dann eine Sicherung hat, die für die Wiederherstellung der Datenbank genutzt werden kann. Während die Datenbank läuft, sind Ihre Dateien permanent geöffnet, und der Zugriff auf sie erfolgt ausschließlich über die Prozesse der Datenbank. Der Benutzer hat keine direkte Kontrolle, wann gelesen und wann geschrieben wird. Das ist der Grund, warum Sie für die Verschlüsselung von Oracle-Datenbankdateien nur Oracle-Methoden einsetzen können.

4.1.1 Transparente Datenverschlüsselung

Transparente Datenverschlüsselung (Transparent Data Encryption, kurz TDE) wurde mit Oracle 10g Release 2 eingeführt. Diese Art der Datenverschlüsselung kann nicht mit früheren Versionen verwendet werden. Um sie zu verwenden, muss der Initialisierungsparameter COMPATIBLE mindestens den Wert10.2.0 haben.

Wie es funktioniert, wird am besten mit einem kleinen Beispiel illustriert: Nehmen wir mal an, Sie möchten die Spalte DNAME in der Tabelle DEPT verschlüsseln. Die Spalte habe folgende Daten:

```
SQL> select dname from dept;

DNAME
--------------
ACCOUNTING
RESEARCH
SALES
OPERATIONS
```

Diese Daten können wir auch direkt aus der Datenbankdatei auslesen. Dazu müssen wir natürlich wissen, um welche Datei es sich handelt, aber die entsprechende Auswahl ist ja begrenzt. Es gibt ja nicht allzu viele Dateien, die hier in Frage kommen. Den physikalischen Speicherort der Daten können Sie mit der folgenden Query, die den Speicherort anhand der Datei- und Blocknummer bestimmt, ermitteln:

Listing 4.1 Physikalischen Speicherort von Daten ermitteln

```
Col file_id for B999
Col block_id for B9999999
Col blocks for B999999
Col file_name for a24
Col value for a5 head bs
Select file_id, block_id, d.blocks, value,
substr(v.name,greatest(length(v.name)-21,1)) file_name
From dba_extents d, v$datafile f, sys.ts$ t, v$parameter p
Where owner=upper('&Owner') and segment_name=upper('&Table') and seg-
ment_type='TABLE' and d.file_id=v.file# and d.tablespace_name = t.name
and p.name='db_block_size'
Order by file_id, block_id;
```

Wenn die Query läuft, geben Sie den Besitzer und den Namen der Tabelle ein und erhalten dann eine Ausgabe, die z.B. so aussehen kann:

```
FILE_ID      BLOCK_ID          BLOCKS     BS   FILE_NAME
-------      --------          -------    ---- ----------------------
4             1037             512        8192 ..oradata/ORCL/users01.dbf
```

Schauen wir uns mal die Daten direkt auf dieser Stufe an. Ich benutze hier auf meinem Laptop einen Hex-Editor, der mir schön die entsprechenden Daten anzeigt (Abbildung 4.1). Unter Unix wäre ein strings-Kommando ausreichend.

Um diese Daten so zu verschlüsseln, dass sie auf dieser Ebene nicht mehr sichtbar sind, muss TDE konfiguriert werden. TDE benutzt ein externes Sicherheitsmodul, das nicht in der Datenbank abgelegt ist: Wenn Sie die Verschlüsselung aktivieren, setzen Sie einen Master Key mit einem Passwort, das in verschlüsselter Form in einem Wallet abgelegt wird.

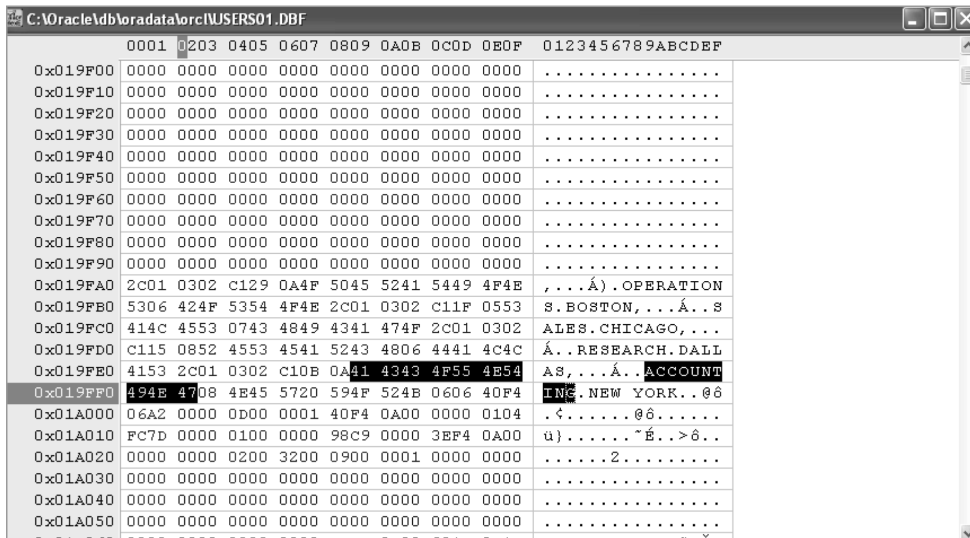

Abbildung 4.1 Dateiinhalt vor Verschlüsselung der Daten

Der Master Key wird auch benutzt, um die Tabellenpasswörter zu verschlüsseln, die im Unterschied zum Master Key in verschlüsselter Form in der Datenbank gespeichert werden. Das Wallet wird zur Erzeugung der Verschlüsselungs-Keys und für die Ver- und Entschlüsselung benutzt.

4.1.1.1 Konfiguration der Verschlüsselung

Die Konfiguration der Verschlüsselung erfolgt in mehreren Schritten. Schritt 1, das Setzen des Master Key, ist nur ein einziges Mal notwendig. Der Master Key kann zwar später erneut gesetzt werden – das kann zum Beispiel notwendig werden, wenn er in die falschen Hände fiel –, aber das erfordert auch eine neue Verschlüsselung der bestehenden Daten.

Das Setzen des Master Key

Der erste Schritt in der Verschlüsselung der Daten besteht darin, der Datenbank mitzuteilen, wo das benötigte Wallet zu finden ist. Das erfolgt über Parameter in der Konfigurationsdatei sqlnet.ora; diese Datei ist standardmäßig im Verzeichnis $ORACLE_HOME/network/admin, beziehungsweise in dem Verzeichnis, auf das die Variable TNS_ADMIN zeigt, zu finden. Sie können die Parameter WALLET_LOCATION oder ENCRYPTION_WALLET_LOCATION verwenden. Oracle empfiehlt Letzteres, und dieser Empfehlung möchte ich mich anschließen. Mit ENCRYPTION_WALLET_LOCATION definieren Sie einen dedizierten Speicherort für das Wallet, das für die transparente Datenverschlüsselung verwendet wird. Das betrachte ich als die bessere Lösung. Der Eintrag in der Datei sqlnet.ora sieht dann zum Beispiel so aus:

```
ENCRYPTION_WALLET_LOCATION =
  (SOURCE =
   (METHOD = FILE)
```

```
    (METHOD_DATA =
     (DIRECTORY = D:\oracle\product\10.2.0\admin\ORCL\wallet)
    )
   )
```

Selbstverständlich muss nach dem Setzen des Speicherorts der SQL*Net Listener (neu) gestartet werden. Dafür verwenden Sie das Kommando: lsnrctl start unter Unix, auf dem PC starten Sie den entsprechenden Service im Control Panel.

Danach kann der Master Key gesetzt werden. Melden Sie sich „AS SYSDBA“ an, danach können Sie das Passwort mit dem Kommando ALTER SYSTEM SET ENCRYPTION KEY setzen:

```
SQL> alter system set encryption key identified by "welcome1";
```

Bitte beachten Sie, dass das Passwort in Anführungsstrichen angegeben wird. Das hat seinen Grund: Dieses Passwort unterscheidet zwischen Groß- und Kleinschreibung!

Dieses Kommando funktioniert übrigens nur, wenn der Speicherort für das Wallet bekannt ist. Ist WALLET_LOCATION/ENCRYPTION_WALLET_LOCATION nicht gesetzt, erhalten Sie den nicht allzu aussagekräftigen Fehler ORA-28368:

```
SQL> alter system set encryption key identified by "welcome1";
alter system set encryption key identified by "welcome"
*
ERROR at line 1:
ORA-28368: cannot auto-create wallet
```

Das Setzen des Parameters in der sqlnet.ora-Datei ist also unumgänglich im ersten Schritt.

Es ist aber nicht zwingend notwendig, dass sich der Benutzer als DBA anmeldet. Das Systemprivileg ALTER SYSTEM ist vollkommen ausreichend, um den Verschlüsselungs-Key zu setzen.

Deshalb gilt: Bei Verwendung der transparenten Datenverschlüsselung muss darauf geachtet werden, dass das Systemprivileg ALTER SYSTEM nur sehr restriktiv vergeben wird, um Missbrauch – auch irrtümlichen – durch unerfahrene Benutzer zu verhindern!

Falls kein Wallet im angegebenen Verzeichnis vorhanden ist, wird es durch das Kommando als verschlüsseltes Wallet angelegt. Die Datei heißt dann – wie immer – ewallet.p12. In diesem Wallet wird der Verschlüsselungs-Key der Datenbank abgelegt. Falls das verschlüsselte Wallet bereits existiert, wird lediglich der Verschlüsselungs-Key der Datenbank erzeugt beziehungsweise neu angelegt. In jedem Fall ist das Wallet nach dem Kommando geöffnet.

Dieses Kommando wird idealerweise nur ein einziges Mal ausgeführt, es sei denn, Sie möchten die Daten erneut mit einem anderen Verschlüsselungs-Key verschlüsseln.

Sie können auch ein existierendes digitales Zertifikat verwenden. Voraussetzung dafür ist, dass es sich um ein X509v3-Zertifikat handelt, das im Key-Usage-Feld Verschlüsselung erlaubt. Wenn Sie OCA verwenden, ist das ohnehin voreingestellt.

4.1.1.2 Das Verschlüsseln der Daten

Nach diesen Vorbereitungen kann die eigentliche Verschlüsselung erfolgen. Dazu muss beim CREATE/ALTER TABLE die Klausel ENCRYPT angegeben werden, hier mal ein einfaches Beispiel:

```
SQL> connect scott/tiger
Connected.
SQL> Alter table dept modify (dname encypt);
Table altered.
```

Hier wurde einfach nur ENCRYPT verwendet, dann wird als Verschlüsselungsalgorithmus AES192 verwendet. Sie können den Verschlüsselungsalgorithmus aber auch ausdrücklich setzen.. Zur Auswahl stehen: 3DES168, AES128, AES192 und AES256. In diesem Fall lautet die Syntax: ENCRYPT USING ' Algorithmus', also beispielsweise ENCRYPT USING 'AES256'.

Standardmäßig wird die Spalte auch mit SALT verschlüsselt. SALT bedeutet, dass vor der eigentlichen Verschlüsselung noch ein zufällig generierter Text an den Klartext angehängt wird. Das ist eine zusätzliche Sicherung insbesondere für Texte mit sich wiederholenden Mustern. SALT hat allerdings einen großen Nachteil: Eine Spalte, die mit SALT verschlüsselt wurde, kann nicht mehr indiziert werden. Soll die Spalte also indiziert werden, muss NO SALT angegeben werden. Hier ein kleines Beispiel zur Illustration, dass dies wirklich so ist:

```
SQL> create index ind_dname on dept(dname);
Index created.

SQL> alter table dept modify (dname encrypt);
alter table dept modify (dname encrypt)
                         *
ERROR at line 1:
ORA-28338: cannot encrypt indexed column(s) with salt

SQL> alter table dept modify (dname encrypt no salt);
Table altered.
```

Sie können auch mehrere Spalten in derselben Tabelle verschlüsseln, allerdings müssen Sie dann immer den gleichen Verschlüsselungsalgorithmus verwenden.[1]

Optional können Sie ein weiteres Passwort bei der Verschlüsselung mit IDENTIFIED BY angeben; dieses Passwort unterscheidet wieder nach Groß- und Kleinschreibung, es sollte also wieder in doppelte Anführungsstriche eingeschlossen werden. Das ist eine zusätzliche Sicherung. Wenn Sie dieses Tabellenpasswort angegeben haben, wird der Verschlüsselungs-Key aus diesem Passwort abgeleitet.

Die wichtigsten Datentypen können so verschlüsselt werden. Die Liste der unterstützen Datentypen umfasst aktuell: CHAR, NCHAR, VARCHAR2, NVARCHAR2, NUMBER,

[1] Bei meinen Tests auf dem PC mit Version 10.2.0.1.0 machte ich übrigens auf Windows XP die Erfahrung, dass nach dem Verschlüsseln einer Spalte auch der Text der übrigen Spalten in der Datenbankdatei nicht mehr (im Hex-Editor) sichtbar war. Ich bin mir nicht sicher, ob ich dieses Verhalten als Bug oder als Feature betrachten soll, auf jeden Fall ist es sehr bequem.

DATE und RAW. Externe Tabellen können auch verschlüsselt werden, allerdings nur, wenn der Tabellentyp ORACLE_DATAPUMP ist. Tabellen, die SYS gehören, können Sie nicht verschlüsseln.

4.1.1.3 Bestimmung des Speicherorts in der Datei

Falls Sie die Daten auf Betriebssystemebene auslesen möchten, bestimmen Sie zuerst – falls Sie nicht bereits wissen, um welche Datei es sich handelt – den Speicherort der Daten. Dies kann zum Beispiel über das am Anfang dieses Kapitels beschriebene Listing 4.1 geschehen.

Der Dump der entsprechenden Blöcke erfolgt dann unter Unix zweckmäßigerweise mit dem Kommando dd. Bei diesem Kommando verwenden Sie die folgenden Parameter:

- if – das Inputfile.
- of – das Outputfile.
- bs – die Blockgröße. Das ist die Oracle-Blockgröße, wie sie im Initialisierungsparameter DB_BLOCK_SIZE festgelegt und in der Query in der Spalte BS ausgegeben wird.
- skip – gibt an, wie viele Blöcke übersprungen werden sollen.
- count – Anzahl der Blöcke, die kopiert werden sollen.

Die dd-Kommandozeile für unser Beispiel würde dann also so aussehen:

```
dd if=users01.dbf of=/tmp/daten_dump bs=8192 skip=1036 count=512
```

Für die Untersuchung der resultierenden Datei genügt dann eine einfache strings/grep-Kommandokombination. Ist der Dump überflüssig – falls zum Beispiel klar ist, wo die Daten zu finden sind –, kann gleich mit strings/grep gesucht werden.

4.1.1.4 Performance der transparenten Datenverschlüsselung

Je nach Größe der Tabelle dauert die Verschlüsselung natürlich unterschiedlich lange, da ja jeder Wert betroffen ist. Um die Dauer der Ver- und Entschlüsselung zu bestimmen, verwende ich die folgende Prozedur, mit der sich die verschiedenen Varianten testen lassen. Das Programmlisting ist relativ lange, aber nicht sehr kompliziert. Es werden ein paar einfache Checks durchgeführt wie beispielsweise, ob der Name der Tabelle oder der Spalte richtig angegeben wurde. Danach wird entweder aus dem Data Dictionary oder über ein „SELECT COUNT(*)“ die Anzahl Rows in der Tabelle ermittelt. Schließlich wird anhand der übergebenen Parameterwerte das eigentliche ALTER TABLE-Kommando dynamisch erzeugt. Die verschiedenen Algorithmen können alle mit und ohne SALT sowie mit und ohne Passwort getestet werden. Es wird sowohl Ver- als auch Entschlüsselung getestet, der entsprechende Zeitbedarf wird über DBMS_UTILITY.GET_TIME und DBMS_UTILITY.GET_CPU_TIME ermittelt. Hier nun ohne weiteren Kommentar die Prozedur:

Listing 4.2 Prozedur zum Testen der Performance der transparenten Datenverschlüsselung

```
create or replace procedure tde_test_proc(towner varchar2 default user,
ttable varchar2, tcol varchar2,
alg varchar2 default 'AES192', salt char default 'Y', pwd char default
'N')
as
cmd varchar2(300) :='';
out_str varchar2(100):='TDE Testvariante:';
ctime1 binary_integer;
ctime2 binary_integer;
ctimetot binary_integer;
etime1 binary_integer;
etime2 binary_integer;
etimetot binary_integer;
alg_v varchar2(10);
salt_v char;
pwd_v char;
exists_obj char;
n_rows number;
begin
-- Test of Benutzer/Tabelle/Spalte richtig angegeben, prueft auf ORA-1403
select 'Y'
into exists_obj
from all_tab_columns
where owner=upper(towner)
and table_name=upper(ttable)
and column_name=upper(tcol);
-- Anzahl Rows ueber Data Dictionary/count(*) ermitteln
cmd:='select to_number(nvl(num_rows,-1)) from all_tables where
owner=upper('||chr(39)||towner;
cmd:=cmd||chr(39)||')  and ta-
ble_name=upper('||chr(39)||ttable||chr(39)||')';
execute immediate cmd into n_rows;
if n_rows = -1 then
  cmd:='select count(*) from '||towner||'.'||ttable;
  execute immediate cmd into n_rows;
end if;
-- Wahl des Algorithmus
alg_v:=upper(alg);
if alg_v not in ('3DES168','AES128','AES192','AES256')
then
  dbms_output.put_line('Fehler - ungueltiger Algorithmus!');
  dbms_output.put_line('Gueltige Algorithmen sind: 3DES168, AES128 ,
AES192, AES256');
  dbms_output.put_line('Standardalgorithmus: AES192');
  return;
end if;
case
  when alg_v = '3DES168' then
    begin
     cmd:=' USING '||chr(39)||'3DES168'||chr(39);
     out_str:=out_str||' 3DES168';
    end;
  when alg_v = 'AES128' then
    begin
     cmd:=' USING '||chr(39)||'AES128'||chr(39);
     out_str:=out_str||' AES128';
    end;
  when alg_v = 'AES192' then
    begin
     cmd:=' USING '||chr(39)||'AES192'||chr(39);
     out_str:=out_str||' AES192';
    end;
  when alg_v = 'AES256' then
    begin
     cmd:=' USING '||chr(39)||'AES256'||chr(39);
     out_str:=out_str||' AES256';
    end;
```

```
end case;
-- Auswahl: mit/ohne Tabellenpasswort
pwd_v:= upper(pwd);
case
when pwd_v = 'Y' then
     begin
     cmd:=cmd||' identified by "Gugua129$$3###aSFdhj12"';
     out_str:=out_str||' - mit Tabellenpasswort';
     end;
when pwd_v = 'N' then
     out_str:=out_str||' - ohne Tabellenpasswort';
else
   begin
     dbms_output.put_line('Fehler: verwenden Sie Y or
N(=Voreinstellung)');
     return;
   end;
end case;
-- Auswahl: Salt/no Salt
salt_v:=upper(salt);
case
when salt_v = 'Y' then
     out_str:=out_str||' - mit SALT';
when salt_v = 'N' then
   begin
     cmd:=cmd||' NO SALT';
     out_str:=out_str||' - ohne SALT';
   end;
else
   begin
     dbms_output.put_line('Fehler: verwenden Sie Y(=Voreinstellung) or
N');
     return;
   end;
end case;
-- Test beginnt
dbms_output.put_line('***************************************************
***********');
dbms_output.put_line(out_str);
dbms_output.put_line('Benutzer        : '||towner);
dbms_output.put_line('Tabelle         : '||ttable);
dbms_output.put_line('Spalte          : '||tcol);
dbms_output.put_line('Anzahl Zeilen   : '||to_char(n_rows));
dbms_output.put_line('Zeitmessungen in 1/100 Sekunden');
dbms_output.put_line('***************************************************
***********');
dbms_output.put_line('Operation  Elapsed  CPU');
dbms_output.put_line('************************************');
for i in 1 .. 2 loop
 if i = 1 then
  -- Teil 1: Verschluesselung
  cmd:='alter table '||towner||'.'||ttable||' modify ('||tcol||' encrypt
'||cmd||')';
 else
  -- Teil 2: Entschluesselung
  cmd:='alter table '||towner||'.'||ttable||' modify ('||tcol||' de-
crypt)';
 end if;
 etime1:=dbms_utility.get_time;
 ctime1:=dbms_utility.get_cpu_time;
 execute immediate cmd;
 etime2:=dbms_utility.get_time;
 ctime2:=dbms_utility.get_cpu_time;
 etimetot:=etime2-etime1;
 ctimetot:=ctime2-ctime1;
 if i = 1 then
  dbms_output.put_line('Encrypt    '||to_char(etimetot)||'
'||to_char(ctimetot));
 else
  dbms_output.put_line('Decrypt    '||to_char(etimetot)||'
```

```
'||to_char(ctimetot));
 end if;
end loop;
exception
  when no_data_found then
    dbms_output.put_line('Fehler - Objekt existiert nicht!');
        dbms_output.put_line('Benutzer: '||towner||' Tabelle:
'||ttable||' Spalte: '||tcol);
  when others then
    dbms_output.put_line('ORA-Fehler: '||SQLCODE);
    dbms_output.put_line('ORA-Fehlermeldung: '||SQLERRM);
end;
/
```

Hier mal ein Beispiel für den Output mit den Standardparameterwerten. Vergessen Sie nicht, vorher SET SERVEROUTPUT ON zu setzen:

```
SQL>  exec tde_test_proc(ttable=>'dept',tcol=>'loc');
**************************************************************
TDE Testvariante: AES192 - ohne Tabellenpasswort - mit SALT
Benutzer        : SCOTT
Tabelle         : dept
Spalte          : loc
Anzahl Zeilen   : 4
Zeitmessungen in 1/100 Sekunden
**************************************************************
Operation  Elapsed  CPU
************************************
Encrypt    5    5
Decrypt    2    2
```

Das geht natürlich noch rasend schnell, 4 Zeilen ist aber auch wirklich keine brauchbare Datenmenge. Wir brauchen also mehr Daten. Hier ein kleiner Trick, den ich mir bei [KYTE2001] abgeschaut habe. Wenn ich schnell mal größere Datenmengen brauche, erzeuge ich sie mir einfach über eine Kopie von DBA_OBJECTS bzw. ALL_OBJECTS und wiederhole INSERT INTO .. SELECT * FROM-Anweisungen; in dieser Data Dictionary View gibt's auch in einer leeren Datenbank genügend Zeilen. Hier im Beispiel haben wir sehr schnell eine Tabelle mit über 400 000 Zeilen erzeugt:

Listing 4.3 Prozedur zum schnellen Erzeugen großer Datenmengen

```
Create table my_objects as select * from sys.dba_objects;
Begin
For I in 1 .. 3 loop
Insert into my_objects select * from my_objects;
Commit;
End loop;
End;
/
```

Jetzt erhalten wir sicher aussagekräftigere Ergebnisse für die verschiedenen Verschlüsselungsalgorithmen. Die Verschlüsselung ohne SALT wurde nur mit dem Standardalgorithmus AES192 durchgeführt. Ein Tabellenpasswort wurde mit AES192 und 3DES168 getestet. Tabelle 4.1 summarisiert die Ergebnisse, die Zeiten sind in Hundertstelsekunden.

Die Ergebnisse zeigen auch, wie wichtig es ist, die CPU-Zeit zu messen und nicht nur Elapsed Time, und das auch und gerade auf dem PC. Der Zeitaufwand für die Ver- und Entschlüsselung richtet sich nach der Komplexität des gewählten Algorithmus, was hier auch schön zu sehen ist. Interessant ist auch das Ergebnis bei Verwendung von SALT: zwar

Tabelle 4.1 Zeitbedarf für transparente Datenverschlüsselung

Operation	Elapsed	CPU	Algorithmus	SALT	Passwort
Encrypt	2801	1805	AES192	Ja	Nein
Decrypt	3251	1470			
Encrypt	3573	2045	3DES168	Ja	Nein
Decrypt	3258	1740			
Encrypt	3905	1620	AES128	Ja	Nein
Decrypt	3283	1524			
Encrypt	4944	1872	AES256	Ja	Nein
Decrypt	4536	1609			
Encrypt	3494	1472	AES192	Nein	Nein
Decrypt	3332	1628			
Encrypt	3197	1889	AES192	Ja	Ja
Decrypt	3132	1564			
Encrypt	3412	2090	3DES168	Ja	Ja
Decrypt	2056	1174			

wird die Verschlüsselung schneller, aber das Entschlüsseln dauert länger! Sehr erfreulich ist die Tatsache, dass die Verwendung eines Tabellenpasswortes die Dauer der Ver- und Entschlüsselung nicht signifikant verlangsamt.

Das Ent- und Verschlüsseln einer Tabelle kann sehr zeitaufwändig sein. Handelt es sich also um eine sehr große Tabelle, die Sie verschlüsseln wollen, testen Sie das Kommando vorher in einer Testumgebung.

4.1.1.5 Transparente Datenverschlüsselung im täglichen Betrieb

Man spricht hier übrigens von transparenter Datenverschlüsselung, weil die Verschlüsselung für die Applikation transparent ist. Um auf die Daten einer verschlüsselten Spalte zuzugreifen, benötigt die Applikation keine zusätzlichen Rechte oder Befehle. Besitzt der Benutzer das SELECT-Privileg für die Tabelle, liefert ihm der SELECT-Befehl die Daten wie gewohnt zurück. Die Ver- und Entschlüsselung wird von Oracle intern im Hintergrund durchgeführt und ist damit für den Benutzer und die Applikation vollkommen transparent.

Die Arbeit mit TDE setzt lediglich voraus, dass das Wallet mit dem Master Key geöffnet ist. Zum Öffnen des Wallets wird das Kommando ALTER SYSTEM SET WALLET OPEN IDENTIFIED BY verwendet. Bitte beachten Sie, dass auch hier das Passwort nach Groß- und Kleinschreibung unterscheidet:

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

Ist das Wallet nicht offen, kann auch nicht auf die verschlüsselte(n) Spalte(n) zugegriffen werden. Der Zugriff auf die übrigen unverschlüsselten Spalten der Tabelle ist aber nach wie vor möglich, wie das folgende Beispiel zeigt:

```
SQL>  select dname from dept;
 select dname from dept
                   *
ERROR at line 1:
ORA-28365: wallet is not open

SQL> select loc from dept;

LOC
-------------
NEW YORK
DALLAS
CHICAGO
BOSTON
```

Das Wallet kann mit dem Kommando ALTER SYSTEM SET WALLET CLOSE dann wieder geschlossen werden, was selbstverständlich auch im laufenden Betrieb möglich ist.

Welche Spalten verschlüsselt sind, sehen Sie im Data Dictionary in DBA_ENCRYPTED_COLUMNS/ALL_ENCRYPTED_COLUMNS/USER_ENCRYPTED_COLUMNS. Sie sehen dort auch, welcher Algorithmus und ob SALT verwendet wurde.

4.1.1.6 Automatisches Aktivieren des Verschlüsselungspasswortes

Beim Shutdown der Datenbank wird das Wallet automatisch geschlossen. Dies bedeutet, dass nach dem Startup der Datenbank ein ALTER SYSTEM SET WALLET OPEN durchgeführt werden muss. Das stellt uns aber vor ein gewisses Problem: der Datenbankstart soll ja möglichst vollautomatisch ablaufen. Das bedeutet, dass man das ALTER SYSTEM SET OPEN in einen STARTUP-Trigger einbauen muss. Dort kann aber jeder, der Zugriff auf DBA_TRIGGERS hat, den Trigger-Quellcode und damit auch das Passwort für die Verschlüsselung auslesen.

Das ist also eine Sicherheitslücke. Sie kann auf mehrere Arten geschlossen werden:

- Das Wallet wird zu einem Wallet mit automatischer Anmeldung gemacht. Dies dürfte die eleganteste Lösung sein. Dazu müssen Sie im Oracle Wallet Manager das Wallet öffnen. Dann gehen Sie im Menü auf „Wallet...“, das Menü geht auf, Sie aktivieren in der Auswahl „Automatische Anmeldung“ (Auto Login) und speichern das Wallet. Danach wird es automatisch geöffnet, wenn die Datenbank gestartet wird.
- Sie speichern das Öffnen des Wallets in binärer Form. Dazu legen Sie eine Prozedur an, die das ALTER SYSTEM SET WALLET OPEN durchführt, und speichern dann den PL/SQL-Quellcode in binärer Form in der Datenbank ab. Im Trigger rufen Sie nun diese Prozedur auf. Das Anlegen der Prozedur in binärer Form kann in Version 10.2 direkt mit DBMS_DDL.CREATE_WRAPPED() geschehen. In früheren Versionen musste dazu das WRAP Utility verwendet werden.
- Sie schränken den Zugriff auf DBA_TRIGGERS/ALL_TRIGGERS so ein, dass nur Sie dort lesen können. Sie können sicher sein: Irgendwann wird jemand kommen, der den Zugriff dort haben möchte und gute Gründe hat, warum. Das ist also keine sehr dauerhafte Lösung.

4.1.1.7 Einschränkungen beim Einsatz von TDE

Es bestehen auch einige Einschränkungen beim Einsatz von transparenter Datenverschlüsselung, die teilweise schon gestreift wurden. Die Einschränkung, dass verschlüsselte Spalten nur mit B*-Indizes verschlüsselt werden können, dürfte in der Praxis keine allzu große Hürde sein. Weit über 90 Prozent aller Indizes in einer Oracle-Datenbank verwenden diesen Typ. Allerdings gibt es durchaus Anwendungen, die hiervon betroffen sind, wie zum Beispiel Applikationen, die Oracle Text verwenden.

Range scans funktionieren auch nicht, wenn die Spalte verschlüsselt ist. Diese Einschränkung dürfte am ehesten bei numerischen Daten zu spüren sein. Dieser Punkt sollte in der Applikation auf alle Fälle überprüft werden. Diese Einschränkung allein kann ein Killerkriterium für die Performance sein.

Unangenehm in der Praxis ist sicher auch die Einschränkung, dass Fremdschüssel nicht verschlüsselt werden können.

Eine Spalte, auf die ein Fremdschlüssel definiert ist, kann nicht verschlüsselt werden!

Auch einige Utitilies, die direkt auf Datenbankdateien zugreifen, sowie der Direct Path SQL*Loader und Export/Import funktionieren nicht mit verschlüsselten Spalten. Data Pump Export/Import kann jedoch – das ist in der aktuellen Dokumentation noch falsch beschrieben – benutzt werden.

Zwar können die meisten Datentypen verschlüsselt werden, aber das gilt nicht für LOB-Daten. Wenn Sie BLOBs oder CLOBs verschlüsseln wollen, müssen Sie das DBMS_CRYPTO Package verwenden.

Bevor Sie TDE einsetzen, klären Sie mit den betroffenen Applikationen, ob und inwieweit sie von den Einschränkungen betroffen sind.

4.1.1.8 Datensicherungen mit transparenter Datenverschlüsselung

Für die Sicherung vom Tabellen mit verschlüsselten Spalten kann, wie bereits oben erwähnt, das Oracle Export Utitility nicht verwendet werden. Wenn Sie es trotzdem versuchen, erhalten Sie einen Fehler:

```
About to export specified tables via Conventional Path ...
Table(T) or Partition(T:P) to be exported: (RETURN to quit) > DEPT

EXP-00107: Feature (COLUMN ENCRYPTION) of column LOC in table SCOTT.DEPT
is not supported. The table will not be exported.
…
```

Um verschlüsselte Spalten zu exportieren, müssen Sie den Data Pump Export verwenden. Wenn Sie expdp ohne weitere Optionen verwenden, werden die Daten allerdings unverschlüsselt in der Dump-Datei abgelegt, wie das folgende Beispiel zeigt:

```
C:\Oracle\db\10.2\BIN>expdp scott/tiger directory=dmpdir dump-
file=expdat.dmp tab
les=dept
…
Processing object type TABLE_EXPORT/TABLE/STATISTICS/TABLE_STATISTICS
. . exported "SCOTT"."DEPT"                              5.656 KB       4
rows
ORA-39173: Encrypted data has been stored unencrypted in dump file set.
Master table "SCOTT"."SYS_EXPORT_TABLE_01" successfully loaded/unloaded
…
```

Bitte beachten Sie aber, dass die Tabellenattribute dadurch nicht geändert werden. Die Spalte ist nach wie vor verschlüsselt in der Datenbank, und der Zugriff auf die Daten erfordert den geöffneten Master Key. Jetzt fragt man sich natürlich, wozu man eine verschlüsselte Spalte unverschlüsselt exportieren soll. Am Anfang des Abschnitts wurde bereits erwähnt, dass es manchmal notwendig ist, den Master Key neu zu setzen, und genau für diesen Fall braucht man den unverschlüsselten Export. Das Vorgehen ist dann wie folgt:

- Unverschlüsselter Export der bestehenden Daten.
- Löschen der bestehenden Daten.
- Schließen des bestehenden Wallet (ALTER SYSTEM SET WALLET CLOSE).
- Neuanlegen des Master Key (ALTER SYSTEM SET ENCRYPTION KEY ...).
- Import der Daten. Die Daten werden beim Import erneut verschlüsselt.

Sie können die Daten in der Dumpdatei aber auch verschlüsselt ablegen, dazu müssen Sie ein Passwort im Parameter ENCRYPTION_PASSWORD mitgeben; das Passwort unterscheidet wieder nach Groß- und Kleinschreibung. Dieses Passwort ist nicht das Verschlüsselungspasswort oder der Master Key. Wenn Sie die Daten später wieder mit impdp zurückspielen wollen, müssen Sie dieses Passwort angeben!

Speichern Sie das ENCRYPTION_PASSWORT an einem sicheren Ort. Daten, die mit einem ENCRYPTION_PASSWORT exportiert wurden, können nur mit diesem Passwort zurückgespielt werden!

Selbstverständlich müssen Sie auch noch das Wallet und das Passwort für den Master Key besitzen, wenn Sie Daten mit verschlüsselten Spalten wieder importieren wollen.

Neben expdp/impdp kennt auch Oracle's RMAN Tabellen mit verschlüsselten Spalten, das ist überhaupt kein Problem. Optional können Sie die Daten bei der Sicherung mit RMAN sogar noch mal, nach dem Motto: „Doppelt genäht, hält besser“ ein zweites Mal verschlüsseln, aber das belastet natürlich die CPU. Mehr zu dieser Thematik im übernächsten Abschnitt.

Die Details zur transparenten Datenverschlüsselung finden Sie in der Oracle-Dokumentation im Kapitel 3 des Advanced Security Administrator's Guide [ADVSEC102].

4.1.2 DBMS_CRYPTO

Neben TDE bietet Oracle noch eine weitere Methode zur Ver- und Entschlüsselung von Daten: das DBMS_CYRPTO Package. Dieses Package existiert eigentlich schon recht lange, ich glaube, es wurde bereits mit Version 8.1.6 eingeführt. Damals hieß es allerdings

noch DBMS_OBFUSCATION_TOOLKIT. Da sich diesen Zungenbrecher kein Mensch merken konnte, beschloss Oracle in Version 10, es umzubenennen und DBMS_CRYPTO einzuführen. Das alte Package DBMS_OBFUSCATION_TOOLKIT existiert aber nach wie vor und kann auch noch verwendet werden, ist aber nicht so mächtig wie DBMS_ CRYPTO. Mit DBMS_OBFUSCATION_TOOLKIT können Sie beispielsweise nur DES und Triple-DES als Verschlüsselungsalgorithmen verwenden. DBMS_CYRPTO ist auch als Ersatz für DBMS_OBFUSCATION_TOOLKIT gedacht, speziell für Neuentwicklungen sollten Sie also unbedingt DBMS_CRYPTO nehmen.

Mit beiden Packages können Sie Daten ver- und entschlüsseln, aber im Unterschied zu TDE geschieht das nicht transparent, sondern muss explizit ausprogrammiert werden. Der Aufwand ist also wesentlich höher als bei der Verwendung von TDE, da ja jeder Zugriff über die entsprechenden Package-Routinen erfolgen muss.

DBMS_CRYPTO unterstützt die Datentypen RAW, BLOB, CLOB und NCLOB; DBMS_ OBFUSCATION_TOOLKIT lediglich RAW und VARCHAR2. Andere Datentypen werden nicht unterstützt, was bedeutet, dass Sie, falls Sie zum Beispiel NUMBER oder DATE verschlüsseln wollen, noch Konvertierungsroutinen bereitstellen müssen, die diese Datentypen entsprechend umwandeln. Das geschieht dann zum großen Teil über Routinen im UTL_RAW Package. Wie solche Routinen programmiert werden können, beschreibt [KNOX2004] recht ausführlich und kompetent. Dort finden Sie auch Angaben zur Performance bei Verwendung von DBMS_CRYPTO, wobei hier natürlich das Gleiche wie bei TDE gilt: Erst testen, dann produktiv einsetzen. Nur so vermeiden Sie – speziell bei zeitkritischen Applikationen – unliebsame Überraschungen

Ich beschränke mich hier auf die wichtigsten Punkte, weil ich TDE für die Methode der Wahl halte. Freilich, wenn Sie beispielsweise CLOB- oder BLOB[2]-Daten verschlüsseln wollen oder nicht mit Version 10.2 arbeiten können, bleibt Ihnen nichts anderes übrig, als DBMS_CRYPTO zu verwenden.

Allerdings stellt DBMS_CRYPTO auch sehr nützliche Routinen für Hashing und MAC zur Verfügung. Dazu später mehr.

Für das Verschlüsseln von LOB-Daten gibt es zwei Prozeduren in DBMS_CRYPTO, die ihr Ergebnis in ein BLOB schreiben. Für das Verschlüsseln von RAW-Daten existiert eine Funktion, die ihr Ergebnis im Datentyp RAW zurückliefert. Für das Entschlüsseln existieren dann analog zur Verschlüsselung die entsprechenden zwei Prozeduren für LOB-Daten und eine Funktion für das Entschlüsseln von RAW-Daten.

Das Illustrationsbeispiel auf der nächsten Seite ist eine leicht abgewandelte Variante des DBMS_CRYPTO-Beispiels in [ORAPAC102], in dem eine VARCHAR2-Spalte aus einer Benutzertabelle gelesen und anschließend ver- und entschlüsselt wird. Hier wird allerdings

---

[2] Auf den ersten Blick erscheint die Verschlüsselung von BLOB-Daten nicht sehr sinnvoll. BLOB-Daten sind ja schon binär, was bedeutet, dass Sie nicht sofort lesbar sind. Andererseits bedeutet binär nicht verschlüsselt, und die meisten binären Formate – wenn Sie jetzt beispielsweise an Graphiken oder die Microsoft Office-Produkte wie Word oder Excel denken – sind bestens dokumentiert und deshalb nicht als sicher zu betrachten.

für die Datenkonvertierung von VARCHAR2 nach RAW das UTL_I18N Package verwendet und nicht UTL_RAW. Die Zeichenkette wird am Bildschirm angezeigt und danach in ein RAW umgewandelt, bevor Sie mit ENCRYPT() verschlüsselt wird. Die verschlüsselte Zeichenkette wird wieder am Bildschirm angezeigt, mit DECRYPT() entschlüsselt und schließlich wird noch die entschlüsselte Zeichenkette dargestellt, die – natürlich – mit der ursprünglichen Zeichenkette übereinstimmt:

Listing 4.4 Beispiel für den Einsatz von DBMS_CRYPTO

```
SQL> DECLARE
  2     output_string      VARCHAR2 (200);
  3     encrypted_raw      RAW (2000);    -- stores encrypted binary text
  4     decrypted_raw      RAW (2000);    -- stores decrypted binary text
  5     num_key_bytes  NUMBER := 256/8; -- key length 256 bits (32 bytes)
  6     key_bytes_raw      RAW (32);   -- stores 256-bit encryption key
  7     encryption_type    PLS_INTEGER :=   -- total encryption type
  8                              DBMS_CRYPTO.ENCRYPT_AES256
  9                            + DBMS_CRYPTO.CHAIN_CBC
 10                            + DBMS_CRYPTO.PAD_PKCS5;
 11     tab_data  varchar2(200);
 12  BEGIN
 13     select mydata into tab_data from crypto_test where rownum=1;
 14     DBMS_OUTPUT.PUT_LINE ( 'Original string:  ' || tab_data);
 15     key_bytes_raw := DBMS_CRYPTO.RANDOMBYTES (num_key_bytes);
 16     encrypted_raw := DBMS_CRYPTO.ENCRYPT
 17        (
 18           src => UTL_I18N.STRING_TO_RAW (tab_data,  'AL32UTF8'),
 19           typ => encryption_type,
 20           key => key_bytes_raw
 21        );
 22     output_string := UTL_I18N.RAW_TO_CHAR (encrypted_raw,
'AL32UTF8');
 23     DBMS_OUTPUT.PUT_LINE ('Encrypted string: ' || output_string);
 24     decrypted_raw := DBMS_CRYPTO.DECRYPT
 25        (
 26           src => encrypted_raw,
 27           typ => encryption_type,
 28           key => key_bytes_raw
 29        );
 30     output_string := UTL_I18N.RAW_TO_CHAR (decrypted_raw,
'AL32UTF8');
 31
 32     DBMS_OUTPUT.PUT_LINE ('Decrypted string: ' || output_string);
 33  END;
 34  /

Original string:  halleluja
Encrypted string: %¿¿Ù"¿¿¿
Decrypted string: halleluja

PL/SQL procedure successfully completed.
```

Bitte beachten Sie auch, dass die verwendete Schlüssellänge in Abhängigkeit vom gewählten Algorithmus 8, 16 oder 32 Bytes betragen muss.

4.1.2.1 Padding

Es können auch unterschiedliche Block-Verschlüsselungsmethoden (Block Ciphers) angegeben werden. Das hat Folgen: Die Größe der resultierenden Daten ist abhängig von der verwendeten Block-Verschlüsselungsmethode. Dabei arbeiten diese Methoden mit festen

Datengrößen, die immer Vielfache von 8 Bytes sind. Daten, die dieser Anforderung nicht genügen, können nicht verschlüsselt und müssen zunächst auf die gewünschte Größe gebracht werden. Dieses Anpassen der Größe durch Auffüllen mit zusätzlichen Bytes wird Padding genannt. Die Zeichenkette 'GUGUS' zum Beispiel muss mit 3 zusätzlichen Bytes aufgefüllt werden, damit sie 8 Bytes beträgt.

Oracle empfiehlt für das Padding PAD_PKCS5, das nach dem PKCS#5-Standard, der auf einem Passwort basiert, auffüllt. Dieser Algorithmus wird auch im Beispiel verwendet. Weitere Varianten sind PAD_ZERO, das mit binären Nullen auffüllt, und PAD_NONE, das aber kein Padding vornimmt. Dann müssen Sie als Programmierer sich darum kümmern, dass die 8-Byte-Grenzen eingehalten werden. Dies bedeutet: Sie machen sich mehr Arbeit als nötig. Einer der großen Vorteile von DBMS_CRYPTO ist ja gerade das automatische Padding, um das Sie sich nicht mehr kümmern müssen. DBMS_OBFUSCATION_TOOLKIT bietet diese Bequemlichkeit nicht, dort müssen Sie sich immer um das Padding kümmern.

4.1.2.2 Speicherung von mit DBMS_CRYPTO-verschlüsselten Daten

CHAR- und VARCHAR2-Daten, die Sie mittels DBMS_CRYPTO verschlüsselt haben, sollten Sie nicht als CHAR oder VARCHAR2 abspeichern. Das Problem hier ist, dass diese Datentypen von Oracle interpretiert werden. Die Interpretation der Daten ist von den sprach- und länderspezifischen Einstellungen (Globalization Support oder National Language Support, kurz NLS) abhängig. Die Daten mögen zwar verschlüsselt sein, doch handelt es sich nach wie vor um eine CHAR- oder VARCHAR2-Spalte, und im Unterschied zu TDE ist die Ver- und Entschlüsselung nicht transparent.

Eine besonders wichtige Rolle bei dieser Interpretation spielt der Zeichensatz der Datenbank. Der verwendete Zeichensatz wird beim CREATE DATABASE angegeben und ist dann mehr oder weniger für alle Zeiten festgelegt. Der Benutzer verwendet normalerweise den gleichen Zeichensatz. Welcher Zeichensatz verwendet werden soll, wird über die Umgebungsvariable NLS_LANG festgelegt.

Die Auswirkungen unterschiedlicher Zeichensätze auf dieselben Daten lässt sich sehr gut mit der SQL-Funktion CONVERT() demonstrieren. Mit der Funktion CONVERT() können Daten in verschiedenen Zeichensätzen dargestellt und umgewandelt werden. Hier im Beispiel wird die verschlüsselte Zeichenkette von vorhin verwendet. Der ursprüngliche Zeichensatz sei US7ASCII, lange die Standardeinstellung bei Oracle und auch noch heute in den USA viel verwendet, als Zielzeichensatz ist der in Europa oft eingesetzte WE8ISO8859P1 angegeben:

```
SQL>  select convert('%¿¿Ù"¿¿¿','WE8ISO8859P1','US7ASCII') from dual;

CONVERT(
--------
%¿¿¿"¿¿¿
```

Wie man sieht, wird die Zeichenkette unter WE8ISO8859P1 anders als unter US7ASCII interpretiert (und dargestellt). Die Entschlüsselung dieser Zeichenkette ergibt dann natür-

lich auch je nach Zeichensatz ein anderes Ergebnis. Bei CLOB-Daten gilt die gleiche Einschränkung, auch hier bestimmen die NLS-Einstellungen die Interpretation der Daten.

CHAR-, VARCHAR2- und CLOB-Daten (die mit DBMS_CRYPTO verschlüsselt werden) werden in Abhängigkeit von den sprach- und länderspezifischen Einstellungen interpretiert. Bei der Speicherung solcher Daten müssen die NLS-Einstellungen immer beibehalten werden!

Um diese Probleme zu vermeiden, empfiehlt es sich, die Daten gleich als RAW oder BLOB abzuspeichern. RAW- und BLOB-Daten werden von Oracle nicht interpretiert und sind damit nicht durch die NLS-Einstellungen bestimmt.

Wie man sich jetzt schon denken kann, beschränkt sich dieses Problem nicht nur auf verschlüsselte Daten. Dazu ein Beispiel mit Umlauten:

```
SQL> select convert('üäö','WE8ISO8859P1','US7ASCII') from dual;
CON
---
¿¿¿
```

Insbesondere beim Transfer von Daten zwischen verschiedenen Datenbanken muss also darauf geachtet werden, dass es nicht zum Datenverlust aufgrund unterschiedlicher Zeichensätze kommt. Das kann insbesondere bei Daten, die mittels Oracle Export und Import transferiert werden, vorkommen. Achten Sie in diesen Fällen immer besonders sorgfältig auf die Einstellungen von NLS_LANG!

4.1.2.3 Hashing und MAC

Einfaches Hashing

Neben den Ver- und Entschüsselungsroutinen bietet das DBMS_CRYPTO Package auch Funktionen, mit denen ein Hashwert gebildet werden kann. Genau wie bei einer Verschlüsselungsfunktion wird eine mathematische Funktion auf einen Klartext angewendet. Als Ergebnis erhalten Sie eine neue – unverständliche – Zeichenkette, wie das folgende Beispiel zeigt:

```
declare
str varchar2(80);
begin
str:=utl_i18n.raw_to_char(dbms_crypto.hash(utl_i18n.string_to_raw('hallel
uja'),dbms_crypto.hash_sh1));
dbms_output.put_line(str);
end;
/
-$°É÷"ùSábè•!D¦5•¬•…

PL/SQL procedure successfully completed.
```

Es muss darauf hingewiesen werden, dass Sie diese Funktion (und auch die MAC-Funktion) nur in einem PL/SQL-Block verwenden können. Hier werden PL/SQL-Variable verwendet, und solche Variablen können nicht direkt in SQL angesprochen werden:

```
SQL> select dbms_crypto.hash(utl_i18n.string_to_raw('halleluja'),
dbms_crypto.hash_sh1
```

```
  2  from dual;
select dbms_crypto.hash(utl_i18n.string_to_raw('halleluja'),
dbms_crypto.hash_sh1)
                                                         *
ERROR at line 1:
ORA-06553: PLS-221: 'HASH_SH1' is not a procedure or is undefined
```

Eine Hashfunktion funktioniert aber im Unterschied zur Verschlüsselung nur in einer Richtung. Im Unterschied zur Verschlüsselung enthält ein Hashwert keinerlei Information über den Ursprungswert. Es gibt keine Unhash-Funktion und somit keine Möglichkeit – außer man ratet – aus dem Hashwert auf den ursprünglichen Wert zurückzuschließen. Eine Hashfunktion ist quasi eine Einbahnstraße.

Von den im DBMS_CRYPTO unterstützten Algorithmen MD4, MD5 und SHA-1 wird von Oracle SHA-1 empfohlen, weil dieser Algorithmus im Unterschied zu den MD-Algorithmen ein wenig resistenter gegenüber Brute-Force-Angriffen ist.

Einsatzgebiete von Hashfunktionen

Hashfunktionen eignen sich zwar nicht zum Verschlüsseln der Daten, sind aber hervorragend zur Prüfung der Datenintegrität geeignet. Sie können so zum Beispiel applikatorische Passworte in einer Tabelle speichern und die Authentisierung des Benutzers überprüfen. Gibt der Benutzer (oder ein Programm) das Passwort ein, wird zuerst der Hashwert berechnet und dann mit dem Hashwert in der Tabelle verglichen.

Selbst wenn ein Angreifer Zugriff auf die Tabelle bekommt, hat er damit noch keine Möglichkeit, die Passwörter zu ermitteln. Wären die Passwörter dagegen verschlüsselt gespeichert, könnte der Angreifer auch die originalen Passwörter bekommen, sobald er den Verschlüsselungsalgorithmus geknackt hat.

Die einfache Hashfunktion hat allerdings den Nachteil, dass man sie durch einen Brute-Force-Angriff erraten kann. Kennt ein Angreifer die verwendete Hashfunktion und die Hashwerte der Applikation, ist es ein Leichtes, ein kleines Programm zu schreiben, das alle möglichen Zeichenketten generiert, die Hashfunktion darauf anwendet und dann das Ergebnis mit dem Hashwert der Applikation vergleicht. Wenn es sich um denselben Hashwert handelt, ist das gesuchte Passwort gefunden. Diese Technik verwenden übrigens auch Oracle-Passwortprüfungsprogramme, mit denen die bekannten Oracle Standardbenutzer und -passwörter ermittelt werden (für einen Überblick hier siehe http://www.petefinnigan.com). Im ersten Kapitel wurde dies bereits im Detail besprochen.

Gesichertes Hashing: die MAC-Funktion

Neben der Hashfunktion bietet DBMS_CRYPTO auch eine MAC-Funktion. MAC steht für Message Authentication Code und stellt eine Erweiterung des einfachen Hashing dar. Im Unterschied zu diesem wird hier die Hashfunktion noch durch einen Schlüssel geschützt. Der Angreifer muss also neben dem verwendeten Algorithmus auch den Schlüssel kennen. Hier können MD5 und SHA-1 verwendet werden.

Ein guter Schlüssel sollte ein Schlüssel sein, der nicht so einfach erraten werden kann. Am besten sind dafür zufällige Zeichenfolgen geeignet, die sich aber mathematisch oft gar

nicht so einfach herstellen lassen. Einen entsprechenden Generator, der zufällige Zeichen erzeugt, liefert Oracle aber im DBMS_CRYPTO Package gleich über die Funktionen RANDOMBYTES, RANDOMINTEGER und RANDOMNUMBER mit. Hier im Beispiel wird RANDOMBYTES verwendet :

Listing 4.5 Beispiel für Hashing mit MAC-Funktion

```
SQL> declare
  2  str varchar2(80);
  3  secret_key raw(16);
  4  begin
  5  secret_key:=dbms_crypto.randombytes(16);
  6
str:=utl_i18n.raw_to_char(dbms_crypto.mac(utl_i18n.string_to_raw('hallelu
ja'),
  7
dbms_crypto.hmac_sh1,secret_key));
  8  dbms_output.put_line(str);
  9  end;
 10  /
–²ÃLc•_Tûû·$ë»2£u!ÍT
```

Ein Wort der Warnung: Wenn Sie zufällige Zeichenfolgen verwenden, müssen Sie diese absolut sicher abspeichern. Falls Sie den Schlüssel verlieren, besteht keine Aussicht, ihn zu rekonstruieren, da er ja auf zufällig erzeugten Zeichen beruht!

4.2 Die Verschlüsselung von Datensicherungen

Die Datensicherung ist ein fester Bestandteil in den wiederkehrenden Abläufen jeder EDV-Abteilung. Unabhängig davon, wann und wie oft sie durchgeführt wird, liegt das Hauptaugenmerk bei ihr auf dem Schutz gegen Datenverlust. Die Datensicherung dient zuerst und vornehmlich zum Schutz gegen einen Verlust der Daten, wobei es in diesem Zusammenhang unerheblich ist, wie die Daten verloren gehen. Das bedeutet: Die Daten müssen effektiv gesichert und im Bedarfsfall schnell und vollständig zurückgespielt beziehungsweise restauriert werden können.

Das ist sicher einer der Hauptgründe, warum eine zusätzliche Verschlüsselung der Sicherungsdaten bisher eher zweitrangig war. Die Verschlüsselung der Sicherung macht den ganzen Ablauf komplexer und auch zeitaufwändiger. Abgesehen davon stellt das Verschlüsseln der Datensicherung eine zusätzliche Belastung der CPU dar.

Dieses Argument wird natürlich hinfällig, wenn die Sicherungsbänder in die falschen Hände geraten. Für manche Firmen – denken Sie zum Beispiel an eine Kreditkartenfirma – ist so etwas ein ziemliches Horrorszenario, das absolut ruinös werden kann. In diesem Falle ist eine zusätzliche Verschlüsselung der Daten das letzte Hindernis, das den Angreifer von einem uneingeschränkten Zugriff auf Ihre Daten trennt.

4.2.1 Verschlüsselung durch die Backup-Software

Kommerziell verfügbare Sicherungssoftware bietet heutzutage auch immer öfter die Möglichkeit, die Datensicherungen zu verschlüsseln. Das ist eine feine Sache und bei mehr oder weniger statischen Daten wie beispielsweise Konfigurations- oder MS Office-Dateien eine gute Lösung.

Diese Art der Verschlüsselung kann speziell im Zusammenhang mit Oracle-Datenbankdateien, aber nur bei so genannten Offline Backups eingesetzt werden. Bei einem Offline Backup wird die Datenbank für die Dauer der Sicherung heruntergefahren. Während der Sicherung ist die Datenbank heruntergefahren, die Benutzer haben keinen Zugriff auf sie, und die Datenbankdateien werden in dieser Zeit auch nicht mehr verändert.

Im Zeitalter des Internet und des allgegenwärtigen Zugriffs rund um die Uhr sind Offline Backups aber eher die Ausnahme. Stattdessen fährt man Online Backups, bei denen die Datenbank während der Sicherung offen und für die Benutzer zugänglich ist. Es ist zwar immer noch eine gute Idee, die Sicherung zu Zeiten, in denen die Daten nur minimal verändert werden, durchzuführen (dies hat vor allem einen Grund darin, dass die Datenbank im Online Backup-Modus mehr Redo generiert, wobei bei der Sicherung mit RMAN das Umschalten in den Backupmodus entfällt), aber zwingend ist das nicht; die Datenbank ist während der Sicherung für die Benutzer uneingeschränkt verfügbar.

Für Daten, die während der Sicherung verändert werden, ist diese Art der Verschlüsselung also ungeeignet; wobei man sich allerdings mit einem – platzfressenden – Trick behelfen kann: In einem ersten Schritt erfolgt die Sicherung auf Festplatte, und dann wird in einem zweiten Schritt diese Sicherung verschlüsselt und auf die eigentlichen Sicherungsbänder gespielt. Das Recovery der Daten erfolgt ebenfalls in zwei Schritten: im ersten Schritt werden die Sicherungsbänder auf die Festplatte kopiert, dort entschlüsselt und erst dann das eigentliche Recovery mit den entschlüsselten Daten durchgeführt.

Es geht aber auch einfacher – allerdings erst seit Oracle Version 10.2.

4.2.2 Verschlüsselung mit RMAN

Oracle 10.2 führte die Möglichkeit ein, Backups, die mit RMAN erstellt wurden, zu verschlüsseln. Mit früheren Versionen geht das leider nicht, der Initialisierungsparameter COMPATIBLE muss mindestens den Wert 10.2.0 haben.

Es können auch nur Backups, die als RMAN Backup Sets erstellt wurden, verschlüsselt werden, mit RMAN Image Copies ist dies nicht möglich.

Für die Verschlüsselung können in Version 10.2 die verschiedenen AES-Algorithmen – also 128Bit, 192Bit und 256 Bit verwendet werden. Die verschiedenen Algorithmen können Sie V$RMAN_ENCRYPTION_ALGORITHMS entnehmen, dort sehen Sie auch, dass AES128 die Standardeinstellung ist:

```
SQL> select  ALGORITHM_NAME, ALGORITHM_DESCRIPTION,IS_DEFAULT,
  2  RESTORE_ONLY  from  v$rman_encryption_algorithms;
```

```
ALGORITHM_NAME       ALGORITHM_DESCRIPTIO DEF RES
-------------------- -------------------- --- ---
AES128               AES 128-bit key      YES NO
AES192               AES 192-bit key      NO  NO
AES256               AES 256-bit key      NO  NO
```

4.2.2.1 Arten der Backupverschlüsselung

Insgesamt stehen 3 Arten für die Verschlüsselung der Sicherung zur Auswahl:

- Passwort-Modus
- Transparenter Modus
- Dualer Modus

Transparenter und dualer Modus setzen ein Verschlüsselungswallet voraus, wie es uns schon im Zusammenhang mit der transparenten Datenverschlüsselung, wie Sie am Anfang dieses Kapitels besprochen wurde, begegnet ist. Ist das Wallet vorhanden, kann die Backup-Verschlüsselung einfach mit dem Kommando CONFIGURE ENCRYPTION eingeschaltet werden:

```
C:\TEMP>c:\oracle\db\10.2\bin\rman target / nocatalog

Recovery Manager: Release 10.2.0.1.0 - Production on Mon Sep 26 10:33:15
2005
Copyright (c) 1982, 2005, Oracle.  All rights reserved.

connected to target database: ORCL (DBID=1095296994)
using target database control file instead of recovery catalog
RMAN> configure encryption for database on;

new RMAN configuration parameters:
CONFIGURE ENCRYPTION FOR DATABASE ON;
new RMAN configuration parameters are successfully stored
```

Mit CONFIGURE ENCRYPTION ändern Sie die Konfiguration des Backups, automatisch werden dann alle folgenden Backups verschlüsselt. Statt der ganzen Datenbank können Sie mit der Klausel FOR TABLESPACE <Tablespace-Name> auch nur einzelne Tablespaces für den verschlüsselten Backup angeben. Ist die Verschlüsselung aktiviert, werden automatisch auch die archivierten Redologs verschlüsselt. Um die Verschlüsselung wieder auszuschalten, geben Sie einfach CONFIGURE ENCRYPTION FOR DATABASE OFF an.

Falls Sie ein Wallet mit automatischer Anmeldung (Auto-Login) verwenden, können Sie Backup und Recovery immer durchführen, weil dieses Wallet ja immer offen ist. Ist dies nicht der Fall, muss das Wallet für den Backup oder das Recovery geöffnet sein. Innerhalb eines RMAN-Skripts müssten Sie dann diese Zeile einbauen:

```
sql 'Alter system set wallet open identified by "…"';
```

Weil Sie hier das Verschlüsselungspasswort für die Datenbank angeben müssen, sollten Sie sehr sorgfältig darauf achten, dass das Skript gut geschützt ist und nicht von unbefugten Personen eingesehen werden kann.

Abgesehen davon sind keine weiteren Änderungen an Ihren Backup-Skripts notwendig. Die Syntax des BACKUP-Kommandos ändert sich überhaupt nicht, wenn der Backup verschlüsselt wird.

Das Recovery der Datenbank aus einem verschlüsselten Backup ist dann vollkommen transparent. Selbstverständlich muss auch das Wallet vorhanden und offen sein, sonst sind keine weiteren Schritte für das Recovery nötig.

Wenn Sie ein Wallet mit automatischer Anmeldung verwenden, sollten Sie, um die Sicherheit nicht zu kompromittieren, darauf achten, dass es unabhängig von der Datenbank gesichert wird. Tun Sie das nicht, braucht der Anwender ja nur das Wallet zusammen mit dem Backup einzuspielen, um uneingeschränkten Zugriff auf die Datenbank zu erhalten. Deshalb sollten Sie auch, falls Sie das Wallet explizit im RMAN-Skript setzen, dafür Sorge tragen, dass das RMAN-Skript nicht zusammen mit der Datenbank abgespeichert wird.

Falls Sie mit dem Wallet arbeiten, kann es auch vorkommen, dass Sie das Passwort dort ändern müssen oder wollen. Für das Recovery ist das übrigens kein Problem. Oracle speichert auch die alten Passwörter im Wallet ab. Wenn Sie also das Passwort von „pass345code" auf „new123pass" ändern mussten und nun ein Recovery mit Daten, die noch mit dem alten Passwort „pass345code" gesichert wurden, durchführen müssen, sollte das überhaupt kein Problem sein.

Das Wallet für die Datenbank muss unabhängig von der Datenbank und absolut zuverlässig gesichert werden. Recovery ohne das Wallet – und auch das Passwort, falls keine automatische Anmeldung verwendet wird – ist nicht möglich!

Das zweite RMAN-Kommando für die Verschlüsselung ist neben CONFIGURE ENCRYPTION das Kommando SET ENCRYPTION. Mit diesem Kommando können Sie neue Einstellungen vornehmen oder Einstellungen, die mit CONFIGURE ENCRYPTION gemacht wurden, überschreiben. Sie brauchen das Kommando auch, um den dualen oder den Passwort-Modus zu konfigurieren.

Dualer Modus

Um den dualen Modus einzustellen, müssen Sie das Kommando SET ENCRYPTION IDENTIFIED BY ... verwenden und ein Passwort für die Sicherung eingeben. Im dualen Modus muss beim Sichern auch das Wallet existieren. Die Bezeichnung dualer Modus bezieht sich nicht so sehr auf die Sicherung, sondern vielmehr auf das Recovery. Sie können das Recovery dann auch auf einer Maschine durchführen, auf der das Wallet gar nicht existiert. Das ist ganz praktisch, wenn Sie die Sicherung zum Beispiel auf einem Testsystem, auf dem das Wallet nicht existiert, einspielen wollen. Dann brauchen Sie auf dem Testsystem nur das Passwort anzugeben.

Im dualen Modus kann die Sicherung sowohl mit dem Wallet als auch mit dem Backup-Passwort zurückgespielt werden.

Falls Sie auch Offline Backups, bei denen die Datenbank während des Backups heruntergefahren ist, einsetzen, können Sie nur Wallets mit automatischer Anmeldung verwenden. Sobald die Datenbank heruntergefahren ist, wird das Wallet ja geschlossen. Das Wallet kann nur geöffnet werden, wenn die Datenbank läuft, wie das folgende Beispiel zeigt:

```
SQL> shutdown immediate
Database closed.
Database dismounted.
ORACLE instance shut down.
SQL> alter system set wallet open identified by "welcome1";
alter system set wallet open identified by "welcome1"
*
ERROR at line 1:
ORA-01034: ORACLE not available
```

Offline Backups können Sie also nur verschlüsseln, wenn Sie im RMAN mit einem Passwort arbeiten. Denken Sie bitte auch daran, dass Sie, falls Sie bereits transparente Datenverschlüsselung verwenden, die zusätzliche Verschlüsselung über RMAN nicht mehr benötigen.

Passwort-Modus

Im Passwort-Modus wird ausschließlich das Passwort verwendet, auch wenn ein Wallet vorhanden ist. Das Kommando lautet in diesem Fall SET ENCRYPTION ON IDENTIFIED BY ... ONLY und unterscheidet sich vom dualen Modus also nur durch das Wörtchen ONLY.

Bitte beachten Sie, dass hier wieder nach Groß- und Kleinschreibung unterschieden wird. Wenn Sie das Passwort nicht mit doppelten Hochkommas umranden, werden Klein- in Großbuchstaben umgewandelt. Die Klausel IDENTIFIED BY Gugus wird also als IDENTIFIED BY GUGUS interpretiert, während bei IDENTIFIED BY „Gugus“ das Wörtchen Gugus, um Groß- und Kleinschreibung zu bewahren, obligatorisch in Hochkommata angegeben werden muss.

4.2.2.2 Auswahl des Verschlüsselungsalgorithmus

Die Auswahl des Verschlüsselungsalgorithmus für Backup und Recovery kann auf zwei Arten vorgenommen werden: entweder über das Kommando CONFIGURE ENCRYPTION USING oder durch SET ENCRYPTION USING. Die erste Variante funktioniert in den ersten 10.2 Releases aber nur scheinbar:

```
RMAN> configure encryption using 'AES256';

new RMAN configuration parameters:
CONFIGURE ENCRYPTION ALGORITHM 'AES256';
new RMAN configuration parameters are successfully stored
```

Es gibt keinen Fehler, RMAN verwendet jedoch immer die Standardeinstellung. Das ist aber kein großes Problem, mit SET ENCRYPTION lässt sich der gewünschte Algorithmus problemlos einstellen.

4.2.2.3 Recovery mit verschlüsselten Backups

Das Recovery von verschlüsselten Backups unterscheidet sich von anderen Recoveries lediglich durch die Angabe des Wallet- bzw. Verschlüsselungspasswortes im Kommando SET DECRYPTION. Wurde ein Wallet verwendet, muss das natürlich auch vorhanden sein. Das ist schon alles, was Sie beachten müssen, ansonsten kann das Recovery wie gewohnt durchgeführt werden.

In RMAN Scripts wird das Passwort beim Recovery mit SET DECRYPTION IDENTIFIED BY ... angegeben werden. Statt eines einzigen können sogar gleich mehrere Passwörter – jeweils durch Kommata getrennt – angegeben werden. RMAN probiert dann einfach alle durch, bis eines gefunden wird, das funktioniert (oder auch nicht).

4.2.2.4 Performance von verschlüsselten Backups

Auf Anhieb würde man erwarten, dass die Performance von der Schlüssellänge des gewählten Algorithmus abhängt. Testen Sie das, aber erwarten Sie hier keine allzu großen Differenzen. Bei meinen zugegebenermaßen recht einfachen Tests, bei denen ich nur das Backup eines einzelnen Tablespace testete, betrug die Differenz zwischen den verschiedenen Algorithmen typischerweise nur um die 10 Sekunden. Interessanterweise war der unverschlüsselte Backup auch nicht wesentlich schneller als der verschlüsselte, und die Backupzeit erhöhte sich auch nicht sehr, wenn Tabellen mit verschlüsselten Spalten im Tablespace zu finden waren.

4.2.2.5 Informationen über die verwendete Backup-Konfiguration

Informationen zum verwendeten Algorithmus erhalten Sie in RMAN mit dem Kommando SHOW ENCRYPTION ALGORITHM. Was alles verschlüsselt wird, können Sie sich mit den Kommandos SHOW ENCRYPTION FOR DATABASE und SHOW ENCRYPTION FOR TABLESPACE anzeigen lassen:

```
RMAN> show encryption algorithm;

RMAN configuration parameters are:
CONFIGURE ENCRYPTION ALGORITHM 'AES256';

RMAN> show encryption for database;

RMAN configuration parameters are:
CONFIGURE ENCRYPTION FOR DATABASE OFF;

RMAN> show encryption for tablespace;

RMAN configuration parameters are:
CONFIGURE ENCRYPTION FOR TABLESPACE 'USERS' ON;
```

Diese Informationen stehen teilweise auch in der Datenbank in der Data Dictionary View V$RMAN_CONFIGURATION:

```
SQL> select name,value from v$rman_configuration order by 1;

NAME                               VALUE
---------------------------------- -------------------------------------
ENCRYPTION ALGORITHM               'AES256'
ENCRYPTION FOR DATABASE            OFF
```

Die Liste der Algorithmen, die Sie für den Backup verwenden können, finden Sie, wie bereits weiter oben erwähnt, in V$RMAN_ENCRYPTION_ALGORITHMS.

Weitere Details zur Verschlüsselung von Backups können Sie in der offiziellen Oracle-Dokumentation in Kapitel 6 [ORABKA102] nachlesen.