ORACLE

ORACLE DOJO NR. 10

HEINZ-WILHELM FABRY

Daten verschlüsseln

Basisschutz für jeden Datentransfer und jeden Datenspeicher

Oracle Dojo ist eine Serie von Heften, die Oracle Deutschland B.V. zu unterschiedlichsten Themen aus der Oracle-Welt herausgibt.

Der Begriff Dojo [‘do:dʒo] kommt aus dem japanischen Kampfsport und bedeutet Übungshalle oder Trainingsraum. Als „Trainingseinheiten“, die unseren Anwendern helfen, ihre Arbeit mit Oracle zu perfektionieren, sollen auch die Oracle Dojos verstanden werden. Ziel ist es, Oracle-Anwendern mit jedem Heft einen schnellen und fundierten Überblick zu einem abgeschlossenen Themengebiet zu bieten.

Im *Oracle Dojo Nr. 10* beschäftigt sich Heinz-Wilhelm Fabry aus der Business Unit Datenbank mit dem Thema Database Security, wobei er den Schwerpunkt auf die unterschiedlichen Möglichkeiten der Verschlüsselung legt. Nach dem Studium dieses Dojo sind Sie in der Lage, einen Basisschutz für jeden Datentransfer und für die Datenbank einzurichten und somit eine sichere Datenhaltung zu gewährleisten.

ORACLE®

Inhalt

1 **Einleitung** *5*

2 **Daten im Transfer verschlüsseln** *7*

2.1 Native Netzwerkverschlüsselung *13*

2.1.1 Integritätsprüfungen *16*

2.2 SSL *16*

2.2.1 Zertifikate einrichten *18*

2.2.2 Konfigurationsdateien des Servers bearbeiten *21*

2.2.3 Konfigurationsdateien des Clients bearbeiten *23*

2.2.4 Benutzer anlegen und testen *25*

3 **Gespeicherte Daten verschlüsseln** *27*

3.1 Prozedural in allen Editionen *27*

3.2 Deklarativ mit Advanced Security Transparent Data Encryption (TDE) in der Enterprise Edition *32*

3.2.1 Einleitung *33*

3.2.2 Vorbereitung: Wallets, Keystores, Passwörter und Schlüssel *34*

3.2.3 Tabellenspalten, SecureFile LOBs und Tablespaces verschlüsseln *42*

3.3 TDE und Datenbanken der Version 12 *45*

4 **Noch mehr Sicherheit für Schlüssel, Wallets und Keystores** *46*
4.1 Installation des OKV *48*
4.2 Konfiguration des OKV *49*
4.2.1 Endpoint einrichten *51*
4.3 Datenbank einrichten *51*
4.4 Verwendung von OKV als Repository *53*
4.4.1 Vorbereitung in OKV *53*
4.4.2 Datenbankserver einrichten *54*
4.4.3 TDE Wallet in OKV sichern *54*
4.5 OKV als HSM nutzen *56*
4.5.1 Datenbankserver einrichten *56*
4.5.2 TDE einrichten *58*
4.6 Berichte und Alerts *59*

5 **Fazit** *60*

6 **Weitere Informationen** *61*

ORACLE DOJO NR. 10

HEINZ-WILHELM FABRY

Daten verschlüsseln

Basisschutz für jeden Datentransfer und jeden Datenspeicher

VORWORT DES HERAUSGEBERS

Die Einstellung in Bezug auf IT-Sicherheit hat in den letzten Jahren einen immensen Wandel durchgemacht. Es ist noch gar nicht so lange her, da meinten auch große deutsche Technologiekonzerne es wäre ausreichend, wenn man die Angestellten und externen Berater schriftlich vergattert, dass sie (Bitteschön) keine Daten stehlen und keine Schadsoftware installieren. Es ist noch gar nicht so lange her, da wurde Sicherheitssoftware lediglich als ein Kostenfaktor betrachtet, der keinerlei Nutzen für die Firma erbringt. Unvorstellbar? Heute ist es tatsächlich kaum mehr vorstellbar, mit welcher Ignoranz teilweise vorgegangen wurde, mit wie viel Unwissenheit die Entscheider Entscheidungen in diesem Bereich getroffen haben. Nicht immer, aber oft mit hohem Risiko für das jeweilige Unternehmen.

Das hat sich vollkommen geändert. Jedem ist heute klar, dass es Cyberangriffe von innen wie von außen gibt. Solche Angriffe sind Realität und keine Hirngespinste pathologischer Angsthasen. Der nichtinvestierte Euro von damals war keine gute Geldanlage. Denn IT-Sicherheit ist weit mehr als die Summe aller Sicherheitsprodukte, die man kauft. Eine sichere IT kann nur dann entstehen, wenn man dies als ständigen Prozess begreift, als Aufgabe, die nicht nur den Security Officer betrifft.

Das vorliegende Dojo beschäftigt sich mit **einem** sehr wichtigen Aspekt aus dem Bereich der Sicherheit von Oracle-Datenbanken: Der Verschlüsselung von Daten in der Datenbank. Dies ist ein zentraler Baustein einer sicheren Datenbank, aber nicht die Lösung per se.

Heinz-Wilhelm Fabry, Senior Leitender Systemberater und Leiter des Fokusthemas Database Security, ist einer unserer erfahrensten Systemberater aus dem Bereich Data Security. Ich freue mich sehr, dass er seine langjährige Erfahrung im Bereich „Verschlüsselung von Daten“ in diesem Dojo#10 so anschaulich dargestellt hat.

Ich wünsche Ihnen viel Spaß beim Lesen und beim Testen.

Ihr Günther Stürner
Vice President Sales Consulting

PS: Wir sind an Ihrer Meinung interessiert. Anregungen, Lob oder Kritik gerne an barbara.frank@oracle.com. Vielen Dank!

1 Einleitung

Eine sichere Datenhaltung basiert immer auf der Absicherung aller Komponenten und Zugriffswege. In der Oracle-Terminologie wird das als *defense in depth* bezeichnet. Neben Zugriffsrechten, Rollen und Audit-Möglichkeiten, die der SQL-Standard ohnehin vorsieht, unterstützen eine ganze Reihe eigener Produkte, Optionen und Features im Datenbankumfeld dieses Konzept. Dazu gehören zum Beispiel:

- Oracle Audit Vault and Database Firewall (AVDF) – Data Warehouse für Audit-Daten und Schutz vor SQL-Injection und anderen nicht autorisierten Zugriffen
- Oracle Database Vault (DV) – Option, die Daten auch vor dem Zugriff durch privilegierte Benutzer schützen kann
- Virtual Private Database (VPD) – Feature der Enterprise Edition für einen differenzierteren Zugriffsschutz auf Datenzeilen, der über die Möglichkeiten des SQL-Standards hinausgeht
- Oracle Key Vault (OKV) – Hardware Security Modul (HSM) und Wallet/Keystore Repository
- Oracle Advanced Security (ASO) – Option, die das Redigieren und Verschlüsseln von Daten ermöglicht

Gerade ASO und die darin enthaltene Möglichkeit, Daten auf Speichermedien zu verschlüsseln, nimmt eine zentrale Rolle in jedem Sicherheitskonzept ein. Denn nur weil für Geheimdienste – und das gilt nicht allein für die amerikanischen Dienste – verschlüsselte Daten kein unüberwindbares Hindernis darstellen, bedeutet das nicht, dass Verschlüsselung überflüssig ist. Ganz im Gegenteil.

Verschlüsselung nach neusten Standards bietet einen hohen Schutz:

- vor *Scriptkiddies*, also vor Jugendlichen, die aus unterschiedlichsten Motiven und häufig ohne nennenswertes Know-how mit im Internet zu findenden grafischen Werkzeugen IT-Systeme angreifen
- vor Hobbyhackern, deren Ziel nicht der Missbrauch von Informationen aus gehackten Systemen ist, sondern die das Hacken als eine Art sportlicher Freizeitbeschäftigung betreiben
- vor Insidern, die aus Neugier oder um sich Vorteile zu verschaffen unberechtigt Daten lesen oder manipulieren
- vor Kleinkriminellen, die auf welchen Wegen auch immer versuchen in Systeme einzudringen, um an Informationen zu gelangen, die sie zu ihrem Vorteil nutzen können
- vor dem organisierten Verbrechen

In Kombination mit weiteren Sicherheitsmaßnahmen organisatorischer und technischer Art ist der Einsatz von Verschlüsselungstechnologien daher nach wie vor eine absolut entscheidende Voraussetzung zum Aufbau sicherer IT-Systeme.

Die Oracle-Datenbank stellt Verschlüsselungsmöglichkeiten in unterschiedlicher Form zur Verfügung: Zum einen ist es möglich, die Übertragung von Daten im Netzwerk nativ oder über SSL zu verschlüsseln. Zum anderen stehen prozedurale und deklarative Möglichkeiten zur Verfügung, gespeicherte Daten zu verschlüsseln.

2 Daten im Transfer verschlüsseln

Anwendungen greifen fast ausschließlich über Netzwerke auf Datenbanken zu. Für potenzielle Angreifer ist deshalb der Netzwerkverkehr ein interessanter und darüber hinaus mit einfachsten Mitteln zu attackierender Bereich eines IT-Systems. Deshalb muss dieser Bereich auch unbedingt geschützt werden.

Bevor die unterschiedlichen Möglichkeiten der Netzwerkverschlüsselung betrachtet werden, ist es wichtig, eine Unterscheidung zu treffen: Es muss nämlich unterschieden

werden zwischen der Übertragung von *Daten zum Aufbau einer Benutzersitzung* und der Übertragung von *Daten einer bestehenden Benutzersitzung*.

- Übertragung von Daten zum Aufbau einer Benutzersitzung (Authentifizierung):

  Alle zur Zeit durch den Oracle-Support unterstützten Datenbankversionen übertragen die Authentifizierungsinformationen *grundsätzlich verschlüsselt*. Die Datenbanken erstellen dabei für jede Benutzersitzung/Session immer wieder einen neuen Schlüssel. Dieser Punkt bedarf also keiner weiteren Betrachtung.

- Übertragung von Daten einer bestehenden Benutzersitzung:

  Nachdem der Benutzer eine Benutzersitzung aufgebaut hat, findet bei allen folgenden Übertragungen *keine Verschlüsselung* mehr statt – es sei denn, man hat entweder seine Zugriffe auf die Datenbank über SQL*Net entsprechend konfiguriert oder man hat sein gesamtes Netzwerk verschlüsselt. Dabei ist die Verschlüsselung des gesamten Netzwerks eine aufwendige und eventuell teure Lösung – und zusätzlich sogar etwas unsicherer. Denn die Verschlüsselung des Netzwerks endet an der Netzwerkkarte und kann auf dem Datenbankserver (lokal) unterlaufen werden, während die Verschlüsselung über SQL*Net erst durch die Datenbank beziehungsweise durch den Client aufgelöst wird. Die Entscheidung, welche Form

der Verschlüsselung genutzt wird, hängt sicherlich vom Bedrohungsszenario ab, aber ohne irgendeine Form der Verschlüsselung des Netzwerks sollte nicht auf Datenbanken zugegriffen werden.

Das folgende Beispiel soll das verdeutlichen. Der Benutzer SCOTT baut eine Benutzersitzung auf und fragt die Tabelle EMP ab. Sowohl die Abfrage als auch das Ergebnis sind über das Netzwerk klar sichtbar. Um das nachzuvollziehen, könnte man einen Netzwerksniffer einsetzen, also ein Tool, das den Netzwerkverkehr mitliest und mittels Filter gezielt nach Datenübertragungen suchen kann. Obwohl solche Tools – auch als grafische Werkzeuge – leicht zugänglich und zum Teil sogar Bestandteil der Basisinstallationen bestimmter Betriebssysteme sind, verbieten viele Firmen zu Recht die Installation oder Nutzung eines solchen *Sniffer* genannten Programms. Stattdessen können aber die Mittel der Oracle-Datenbanksoftware genutzt werden, um einen Einblick in den Netzwerkverkehr von und zu einer Datenbank zu erhalten. Dazu muss nur in der Datei SQLNET.ORA ein Tracing mit geeigneten Optionen eingestellt werden, hier:

```
trace_level_client = 16
```

Jede neue Datenbanksitzung wird ohne Neustart der Datenbank ab sofort eine Tracedatei erzeugen. In dieser Tracedatei könnte man dann zum Beispiel folgende Einträge finden:

```
2014-12-04 16:13:41.555617 : nsbasic_bsd:00 00 00 00 00 11 73 65  |......se|
2014-12-04 16:13:41.555623 : nsbasic_bsd:6C 65 63 74 20 2A 20 66  |lect.*.f|
2014-12-04 16:13:41.555628 : nsbasic_bsd:72 6F 6D 20 65 6D 70 01  |rom.emp.|
```

Abb.1: Auszug aus der Tracedatei – unverschlüsseltes SQL

Man sieht in Abbildung 1 genau, welches SELECT-Statement vom Client an den Datenbankserver geschickt wird. Das sieht auf den ersten Blick nicht wirklich gefährlich aus. Dennoch enthalten Datenbankabfragen zum Beispiel in WHERE-Bedingungen oft interessante Informationen für nicht autorisierte Personen.

Aber auch das Ergebnis, das von der Datenbank an den Client geschickt wird, ist in der Tracedatei nachvollziehbar. Abbildung 2 zeigt einen Ausschnitt aus den Informationen zum Mitarbeiter KING:

```
2014-12-04 16:13:41.570604 : nsbasic_brc:28 04 4B 49 4E 47 09 50  |(.KING.P|
2014-12-04 16:13:41.570609 : nsbasic_brc:52 45 53 49 44 45 4E 54  |RESIDENT|
2014-12-04 16:13:41.570615 : nsbasic_brc:FF 07 77 B5 0B 11 01 01  |..w.....|
```

Abb.2: Auszug aus der Tracedatei – unverschlüsselte Benutzerdaten

Ein Angreifer kann also auch alle unverschlüsselt an den Client geschickten Daten abfangen und lesen. Aber auch das ist noch nicht alles: Wenn ein Datenbankbenutzer sein Passwort mit dem Befehl ALTER USER ändert und die

Anwendung dabei eine Netzwerkverbindung benutzt, wird auch das neue Passwort unverschlüsselt übertragen. In SQL*Plus ist deshalb schon seit einigen Jahren der Befehl PASSWORD eingeführt worden, der eine verschlüsselte Übertragung der Passwortänderung ermöglicht.

Wäre das Netzwerk über SQL*Net verschlüsselt, könnten die Informationen aus der Tracedatei für die Abfrage und die Benutzerdaten etwa so aussehen, wie es die Abbildungen 3 und 4 zeigen:

```
2014-12-04 16:24:57.709811 : nspsend:01 FD A0 1A 12 12 7A 13  |......z.|
2014-12-04 16:24:57.709816 : nspsend:68 0C 0B 00 19 6D B0 07  |h....m..|
2014-12-04 16:24:57.709821 : nspsend:AD 79 DA 4C F0 60 48 8B  |.y.L.`H.|
```

Abb. 3: Auszug aus der Tracedatei – verschlüsseltes SQL

```
2014-12-04 16:24:57.715715 : nspsend:E7 4E 1E 36 70 DD 73 7D  |.N.6p.s}|
2014-12-04 16:24:57.715720 : nspsend:C2 6C 18 4C DA E9 EA 53  |.l.L...S|
2014-12-04 16:24:57.715726 : nspsend:18 16 0F 36 C2 E0 3C A0  |...6..<.|
```

Abb. 4: Auszug aus der Tracedatei – verschlüsselte Benutzerdaten

Dieses einfache Beispiel belegt die Bedeutung, die der Verschlüsselung des Netzwerkverkehrs eingeräumt werden *muss*. Das gilt umso mehr, da viele Angriffe auf IT-Systeme innerhalb der von Firewalls geschützten Bereiche durch sogenannte Innentäter erfolgen. Eine Verschlüsselung des

gesamten Netzwerkverkehrs wäre für die meisten Unternehmen allerdings nicht zwingend notwendig und wohl auch zu teuer. Deshalb bietet Oracle die Verschlüsselung und Sicherung der Netzwerkkommunikation zwischen Oracle-Datenbanken und Oracle-Clients an. Dabei ist der Begriff „Oracle-Client" weit zu fassen: Aus der Sicht einer Datenbank ist ein Application-Server oder eine andere Datenbank, die über einen Datenbank-Link zugreift, ebenfalls ein Oracle-Client.

Gerade dieser Schutz ist auch seit der Freigabe der Oracle Datenbank 12c Release 1 Ende Juni 2013 deutlich leichter geworden. Denn sowohl die native Verschlüsselung über SQL*Net als auch die Verschlüsselung über SSL sind seitdem *nicht mehr* als Teil der Advanced Security Option (ASO) zu lizenzieren, sondern stehen – ebenso wie die starken Authentifizierungsmethoden – als Feature der Datenbank sowohl in der Enterprise Edition als auch in der Standard Edition der Datenbank ohne zusätzliche Kosten zur Verfügung. Das gilt auch für vorangegangene Releases, sofern diese noch durch den Support unterstützt werden (zum Beispiel für Oracle Database 11g) und sofern die Software das technisch erlaubt.

2.1 NATIVE NETZWERKVERSCHLÜSSELUNG

Die Netzwerkverschlüsselung über SQL*Net kann innerhalb von Minuten und ohne jede Betriebsunterbrechung aktiviert werden. Nach der Aktivierung werden alle neu aufgebauten Verbindungen automatisch verschlüsselt. Das gilt für alle OCI-basierenden Anwendungen, für *Java-Thick-Client-Anwendungen* und für die meisten neueren *Java-Thin-Client-Anwendungen*. Um dennoch Probleme zu vermeiden, empfiehlt es sich für produktive Umgebungen, die Auswirkungen der Umstellung zunächst auf Testsystemen unter realistischen Bedingungen zu überprüfen.

Die Netzwerkverschlüsselung wird über Einträge in der Datei SQLNET.ORA aktiviert. Die dazu nötigen Einstellungen können einfach in der grafischen Oberfläche des Oracle Net Manager (NETMGR) vorgenommen werden. In Testumgebungen steht allerdings nicht immer eine grafische Oberfläche zur Verfügung. Deshalb werden die Einstellungen hier in der Linemode-Variante vorgestellt.

Zwei Aspekte werden konfiguriert:

- Soll überhaupt verschlüsselt werden?
- Welcher Algorithmus soll verwendet werden?

Ob verschlüsselt wird, legen die Parameter SQLNET.ENCRYPTION_CLIENT beziehungsweise SQLNET.ENCRYPTION_SERVER fest. Dabei hat man die Wahl zwischen vier Einstellungen:

- REJECTED – Verschlüsselung wird grundsätzlich abgelehnt
- ACCEPTED (Default) – Verschlüsselung wird akzeptiert, wenn der Kommunikationspartner das möchte
- REQUESTED – Verschlüsselung wird gewünscht, aber nicht verlangt
- REQUIRED – Verschlüsselung wird verlangt

Entsprechend der Einstellungen auf beiden Seiten ergibt sich nun, ob verschlüsselt wird oder nicht, beziehungsweise ob überhaupt eine Datenbankverbindung zustande kommt. Per Default gilt für beide Seiten der Wert ACCEPTED. Das hat zur Folge, dass beide Seiten für eine Verschlüsselung zur Verfügung stehen, sie diese Verschlüsselung aber nicht ausdrücklich wünschen. Deshalb findet als Default keine Verschlüsselung statt. Die folgende Abbildung zeigt die möglichen Kombinationen und ihre Auswirkungen auf die Verschlüsselung beziehungsweise auf den Verbindungsaufbau zwischen den Kommunikationspartnern.

		CLIENT			
		REJECTED	ACCEPTED	REQUESTED	REQUIRED
SERVER	REJECTED	aus	aus	aus	keine Verbindung
	ACCEPTED	aus	aus*	ein	ein
	REQUESTED	aus	ein	ein	ein
	REQUIRED	keine Verbindung	ein	ein	ein

* *Default ist Accepted (Verschlüsselung und Prüfsummenverfahren sind ausgeschaltet)*

Abb. 5: Zusammenspiel der Client- und Serverparametereinstellungen

Als Nächstes kann noch der Verschlüsselungsalgorithmus gewählt werden. Je nach Version der Datenbank beziehungsweise des Oracle-Clients stehen verschiedene Algorithmen zur Verfügung. Für ältere Datenbankversionen sind die verfügbaren Algorithmen noch im Handbuch „Advanced Security Administrator's Guide" beschrieben. Für Oracle Database 12c erläutert das Handbuch „Oracle Database Security Guide" die Algorithmen. Wird kein Algorithmus festgelegt, stimmen Client und Server beim Aufbau der Verbindung (*handshake*) ab, welcher Algorithmus benutzt wird.

Die Datei SQLNET.ORA auf der Datenbank-Seite könnte etwa um folgende Einträge ergänzt werden:

```
-- allein der erste Eintrag würde schon eine verschlüsselte Kommunikation
-- erzwingen, wäre also quasi eine Art Minimalanforderung: Laut Matrix
-- oben wird entweder verschlüsselt, oder es kommt keine Verbindung zustande

sqlnet.encryption_server                    =   required
sqlnet.encryption_types_server              =   AES256
sqlnet.encryption_client                    =   required
sqlnet.encryption_types_client              =   AES256
```

Die CLIENT-Parameter werden angegeben, da eine Datenbank – wie oben bereits erwähnt – auch als Client agieren kann. Die gleiche Datei kann ebenfalls auf der Client-Seite verwendet werden, vorausgesetzt der Client unterstützt die angegebenen Algorithmen. Die Tatsache, dass datenbankseitige Parameter spezifiziert sind, stört dabei nur den Ästheten.

2.1.1 INTEGRITÄTSPRÜFUNGEN

Die Verschlüsselung allein reicht allerdings nicht aus, um den Netzwerkverkehr abzusichern, denn verschlüsselte Datenpakete können umgeleitet und in veränderter Form wieder eingespielt werden. Um sich dagegen zu schützen, sollten Prüfsummenverfahren genutzt werden. Zur Verfügung stehen hier je nach Datenbankversion unterschiedliche Algorithmen, deren Verwendung ebenfalls über zwei Parameter in den SQLNET.ORA-Dateien von Client und Server konfiguriert wird. Wieder wird festgelegt, ob Prüfsummen überhaupt verwendet werden sollen und eventuell welcher Algorithmus zum Einsatz kommen soll.

```
sqlnet.crypto_checksum_server                = required
sqlnet.crypto_checksum _types_server         = SHA512
sqlnet.crypto_checksum_client                = required
sqlnet.crypto_checksum _types_client         = SHA512
```

2.2 SSL

SSL stellt die Alternative zur nativen Verschlüsselung über SQL*Net dar. Die Beschaffung der für die Arbeit mit SSL notwendigen Zertifikate, ihre Verteilung und ihre Verwaltung erfordern einen gewissen Aufwand. Viele Unternehmen scheuen diesen Aufwand, wenn er sich nur auf einen relativ kleinen Ausschnitt der eigenen EDV bezieht. Und natürlich spielt auch die eigene Erfahrung eine Rolle: Wer unsicher im Umgang mit SSL

ist, wird SSL nicht nutzen. Deshalb wird die SSL-Verschlüsselung im Oracle-Umfeld fast ausschließlich von Unternehmen eingesetzt, die auch in anderen Bereichen ihrer EDV bevorzugt damit arbeiten.

Es stellt sich aber selbstverständlich die Frage, warum dann überhaupt SSL eingesetzt wird. Die Erklärung liegt im zweifachen Nutzen von SSL: Denn erstens kann man damit den Netzwerkverkehr verschlüsseln, aber zweitens kann man unabhängig davon eine eindeutige Identifizierung von Benutzern erreichen. Das wiederum bedeutet einen zusätzlichen Gewinn an Sicherheit vor sogenannten *Man-in-the-Middle-Angriffen*. Wie zum Beispiel im Handbuch „Oracle Database Security Guide" für Oracle Database 12c beschrieben, bietet die native Netzwerkverschlüsselung über das sogenannte *authentication key-fold in* zwar einen hohen Schutz gegen solche Angriffe, aber SSL bietet dagegen den wohl zur Zeit optimalen Schutz. Ist also ein *Man-in-the-Middle-Angriff* ein realistisch eingeschätztes Risiko im selbst ermittelten Bedrohungsszenario, sollte der Einsatz von SSL unbedingt in Betracht gezogen werden. Allerdings muss auch berücksichtigt werden, dass der Verbindungsaufbau über SSL etwas aufwendiger ist als der Aufbau einer nativ verschlüsselten Verbindung und deshalb auch entsprechend langsamer vonstattengeht.

2.2.1 ZERTIFIKATE EINRICHTEN

Wie schon angemerkt arbeitet SSL mit Zertifikaten. Die Zertifikate werden von *Certificate Authorities* ausgestellt. Zum Testen kann man allerdings auch mit selbst signierten Zertifikaten arbeiten. Im Einzelnen sind folgende Arbeitsschritte abzuarbeiten:

Da Zertifikate in Wallets gespeichert werden, wird als erstes ein Wallet auf der Server-Seite angelegt. Oracle bietet dazu drei Werkzeuge an: *MKSTORE, ORAPKI* und den *Oracle Wallet Manager (OWM).* Nur ORAPKI unterstützt alle weiteren durchzuführenden Schritte. Damit nicht ständig zwischen unterschiedlichen Werkzeugen gewechselt werden muss, soll deshalb hier ORAPKI genutzt werden. Folgender Aufruf erzeugt das Wallet:

```
orapki wallet create -wallet . -auto_login -pwd oracle_1
```

Nach dem Parameter -WALLET erfolgt die Angabe des Verzeichnisses, in dem das Wallet angelegt wird, hier also im aktuellen Verzeichnis. Als aktuelles Verzeichnis wird für dieses Dojo „$ORACLE_HOME/network/admin“ gewählt, da hier weitere Dateien liegen, die später noch zu bearbeiten sind. Der Parameter -AUTO_LOGIN bewirkt, dass das Wallet ständig für den Zugriff durch Oracle geöffnet ist. Dazu wird von ORAPKI auf der Betriebssystemebene nicht nur das eigentliche Wallet namens EWALLET.P12, sondern eine weitere

Datei namens CWALLET.SSO angelegt. Normalerweise gibt die sicherheitsverantwortliche Person das Passwort wohl nicht sichtbar ein. Sie verwendet den Parameter -PWD dann einfach nicht und wird deshalb vor dem Ausführen des Befehls nach einem Passwort gefragt. Genügt das Passwort nicht den Sicherheitsanforderungen, erhält man die Fehlermeldung „PKI-01002: Ungültiges Kennwort: Kennwörter müssen mindestens acht Zeichen umfassen und eine Kombination von Buchstaben und Zahlen oder Sonderzeichen enthalten“.

Im nächsten Schritt wird ein selbst signiertes Zertifikat erzeugt und in dem Wallet gespeichert. In diesem Fall wird zusätzlich automatisch ein sogenanntes User-Zertifikat erzeugt. Es wird ebenfalls automatisch in dem Wallet gespeichert.

```
-- Falls der Befehl abgetippt wird, den Zeilenumbruch nicht übernehmen
orapki wallet add -wallet . -dn 'cn=orcl' -keysize 1024
-self_signed -validity 4000 -pwd oracle_1
```

Wie man sieht, ist ein sogenannter *distinguished name* (-DN) anzugeben. Für diesen Artikel genügt es, lediglich den sogenannten *common name* (-CN) anzugeben, hier „ORCL“. Es sind auch weitere Angaben möglich. Außerdem wird die Länge des Schlüssels (512, 1024 oder 2048 Bits) festgelegt, dass es sich um ein selbst signiertes Zertifikat handelt, dass das Zertifikat 4000 Tage lang gültig ist und selbstverständlich

auch das Passwort für das Wallet (entweder hier oder nach der Eingabeaufforderung).

Das auf der Server-Seite gespeicherte Zertifikat wird später auch auf der Client-Seite benötigt. Deshalb wird es mit folgendem Befehl aus dem Wallet in die Datei SERVER.CERT im aktuellen Verzeichnis exportiert.

```
-- Falls der Befehl abgetippt wird, den Zeilenumbruch nicht übernehmen
orapki wallet export -wallet . -dn 'cn=orcl' -cert server.cert
-pwd oracle_1
```

Auf der Client-Seite werden diese Schritte nun analog ausgeführt. Nach dem Anlegen des Wallets wird das Zertifikat erzeugt – natürlich nicht mit dem gleichen *common name*, denn schließlich soll dieses Zertifikat einen Benutzer auf der Client-Seite eindeutig identifizieren. Hier wird der *common name* DOJODBA verwendet. Anschließend wird das Zertifikat ebenfalls in eine Datei exportiert, nämlich in die Datei CLIENT.CERT. Damit kann es im nächsten Schritt der Server-Seite zugänglich gemacht werden. Die Befehle sehen folgendermaßen aus:

```
orapki wallet create -wallet . -auto_login -pwd oracle_1
-- Falls der Befehl abgetippt wird, den Zeilenumbruch nicht übernehmen
orapki wallet add -wallet . -dn 'cn=dojodba' -keysize 1024
-self_signed -validity 4000 -pwd oracle_1
```

```
-- Falls der Befehl abgetippt wird, den Zeilenumbruch nicht übernehmen
orapki wallet export -wallet . -dn 'cn=dojodba' -cert client.
cert -pwd oracle_1
```

Nun werden die Zertifikate vom Server zum Client und vom Client zum Server kopiert (zum Beispiel mit SCP) und in die jeweiligen Wallets importiert:

```
-- Server-Seite
orapki wallet add -wallet . -trusted_cert -cert client.cert
-pwd oracle_1

-- Client-Seite
orapki wallet add -wallet . -trusted_cert -cert server.cert
-pwd oracle_1
```

Damit sind die Schritte abgeschlossen, die in erster Linie mit den Zertifikaten und Wallets zu tun haben.

2.2.2 KONFIGURATIONSDATEIEN DES SERVERS BEARBEITEN

In den nächsten beiden Arbeitsschritten muss die Voraussetzung geschaffen werden, dass der Server das Wallet und die darin abgelegten Zertifikate nutzt. Dazu sind Einträge in den Dateien SQLNET.ORA und LISTENER.ORA nötig.

In der Datei SQLNET.ORA auf der Server-Seite werden folgende Einträge gemacht:

```
-- TCPS, damit die Authentifizierung über SSL genutzt werden kann
SQLNET.AUTHENTICATION_SERVICES = (BEQ,TCPS)

-- Damit authentifiziert die Datenbank Benutzer über SSL
SSL_CLIENT_AUTHENTICATION=TRUE

-- Pfad zum Wallet mit den Zertifikaten
WALLET_LOCATION=     (SOURCE=
                     (METHOD=FILE)
                     (METHOD_DATA=
                     (DIRECTORY=$ORACLE_HOME/network/admin)))
```

Auch die Datei LISTENER.ORA muss angepasst werden. Dazu wird der Parameter WALLET_LOCATION genauso eingestellt wie schon in der Datei SQLNET.ORA. Außerdem wird als Protokoll TCPS angegeben. Der gemeinhin verwendete Port ist hier 2484. Allerdings kann – gerade zum Testen – auch eine andere Portnummer verwendet werden (der gültige Wertebereich geht von 1024 bis 65535). Folgende Einträge sind in der hier verwendeten LISTENER.ORA zu finden:

```
-- TCPS mit Port 2484 für die SSL Verbindungen hinzufügen
LISTENER =
    (DESCRIPTION_LIST =
        (DESCRIPTION =
            (ADDRESS = (PROTOCOL = IPC)(KEY = EXTPROC1521))
            (ADDRESS = (PROTOCOL = TCP)(HOST = 192.168.56.101)
            (PORT = 1521))
```

```
        (ADDRESS = (PROTOCOL = TCPS)(HOST = 192.168.56.101)
        (PORT = 2484))

    )
        )

-- Speicherort des Wallets angeben (analog zur Datei SQLNET.ORA)
WALLET_LOCATION=      (SOURCE=
                      (METHOD=FILE)
                      (METHOD_DATA=
                      (DIRECTORY=$ORACLE_HOME/network/admin)))

-- Nicht der Listener identifiziert den Benutzer, sondern die Datenbank.
SSL_CLIENT_AUTHENTICATION=FALSE
```

2.2.3 KONFIGURATIONSDATEIEN DES CLIENTS BEARBEITEN

In den nächsten Schritten muss die Voraussetzung geschaffen werden, dass der Client das Wallet und die darin abgelegten Zertifikate nutzt. Dazu sind Einträge in den Dateien SQLNET.ORA und TNSNAMES.ORA nötig.

In der Datei SQLNET.ORA werden folgende Einträge gemacht:

```
-- TCPS, damit die Authentifizierung über SSL genutzt werden kann.
SQLNET.AUTHENTICATION_SERVICES= (BEQ, TCPS)
```

```
-- Damit authentifiziert die Datenbank die/den Benutzer über SSL
SSL_CLIENT_AUTHENTICATION=TRUE

-- Speicherort des Wallets auf dem Client
WALLET_LOCATION=     (SOURCE=
                     (METHOD=FILE)
                     (METHOD_DATA=
                     (DIRECTORY = /oracle/product/dbhome/
                                  network/admin)))

/* Der distinguished name des Servers muss identisch sein mit dem Servicenamen
und muss mit dem SSL_SERVER_CERT_DN in der Datei TNSAMES.ORA übereinstimmen. */

SSL_SERVER_DN_MATCH=ON
```

Die Datei TNSNAMES.ORA enthält folgende Einträge:

```
ORCL =
    (DESCRIPTION =
        (ADDRESS_LIST =
            (ADDRESS = (PROTOCOL = TCPS)(HOST = 192.168.56.101)
(PORT = 2484))
        )
        (CONNECT_DATA =
            (SERVER = DEDICATED)
            (SERVICE_NAME = orcl.de.oracle.com)
        )
        (security=(SSL_SERVER_CERT_DN=”cn=orcl”))
    )
```

Hinweis: Wenn die Einträge in den Konfigurationsdateien mit Editoren wie *vi* bearbeitet werden und das zu merkwürdigen Fehlermeldungen führt, bitte die Angaben zu den einzelnen Parametern in *eine* Zeile bringen oder mit Oracle-Tools wie dem bereits erwähnten Oracle Net Manager (NETMGR) die Einträge bearbeiten. Die Konfigurationsdateien von SQL*Net sind zum Teil ausgesprochen „sensibel“, wenn es um Zeilenumbrüche und Leerzeichen geht.

2.2.4 BENUTZER ANLEGEN UND TESTEN

Um die bis hierhin erstellte Beispielkonfiguration zu testen, wird nun ein Benutzer angelegt, der über das im Wallet gespeicherte Client-Benutzerzertifikat authentifiziert wird. Außerdem werden diesem Benutzer die Privilegien CREATE SESSION und die Rolle DBA zugewiesen.

```
CREATE USER dojodba IDENTIFIED EXTERNALLY AS ‘CN=dojodba’;
GRANT create session TO dojodba;
```

Nun kann die SSL-Verbindung vom Client zur Datenbank auf zweifache Weise genutzt werden. Zunächst die Variante, in der nur auf die Netzwerkverschlüsselung zurückgegriffen wird: SCOTT ruft SQL*Plus auf und erhält dann den Prompt zu seinem Passwort. Der zweite Aufruf zeigt, dass DOJODBA sich ohne Passwort bei der Datenbank anmelden

kann: DOJODBA wird über das Zertifikat im Wallet eindeutig identifiziert. Und außerdem wird natürlich auch hier eine verschlüsselte Verbindung genutzt.

```
-- Als Benutzer SCOTT
sqlplus scott@orcl
...
Enter password:
...
SQL> show user
USER is “SCOTT”


-- Als Benutzer DOJODBA
sqlplus /@orcl
...
SQL> show user
USER is “DOJODBA”

...
```

3 Gespeicherte Daten verschlüsseln

3.1 PROZEDURAL IN ALLEN EDITIONEN

Obwohl es nach wie vor zum Lieferumfang der Datenbank gehört, sollte zur prozeduralen Verschlüsselung nicht das inzwischen veraltete Package DBMS_OBFUSCATION_TOOLKIT verwendet werden, sondern das Package DBMS_CRYPTO. Voraussetzung für seine Verwendung ist die Ausführungsberechtigung, die natürlich über ein GRANT EXECUTE erteilt wird.

In konkreten Projekten betrifft die wichtigste Entscheidung für das Arbeiten mit DBMS_CRYPTO den Umgang mit dem Schlüssel oder den Schlüsseln für die Verschlüsselung. Drei Vorgehensweisen sind möglich:

- Der oder die Schlüssel werden in der Datenbank gespeichert. Denkbar wäre, den oder die Schlüssel in einer separaten Spalte der Tabelle abzulegen, die auch die zu verschlüsselnden Daten enthält. Die Speicherung kann alternativ in einer eigenen Tabelle erfolgen. So ist es möglich, Schlüssel zu verwenden, die für unterschiedliche Zeiträume gelten oder die für jeden Datensatz unterschiedlich sind. Für beide Fälle muss natürlich die passende Form der Verknüpfung gewählt werden –

also zum Beispiel eine Verknüpfung über ein in dem Datensatz enthaltenes Datumsfeld oder eine Primär-Fremdschlüsselverknüpfung mit der Datentabelle.

- Problematisch bei diesen Vorgehensweisen ist, dass privilegierten Benutzern, also zum Beispiel Datenbankadministratoren, der Zugriff auf diese Schlüssel und damit auf die verschlüsselten Daten nicht zu entziehen ist.
- Der oder die Schlüssel werden in einer Betriebssystemdatei abgelegt, die im Rahmen der Verschlüsselung beziehungsweise Entschlüsselung gelesen wird. Auch hier liegt die Problematik in der Absicherung der Schlüssel gegen unbefugte Zugriffe, denn privilegierte Betriebssystemnutzer könnten die Datei jederzeit lesen.
- Schließlich können die Schlüssel durch die Benutzer selbst verwaltet werden. Hier besteht allerdings die Gefahr, dass Schlüssel entweder vergessen werden – mit der Folge, dass der Zugriff auf die verschlüsselten Daten endgültig verloren geht – oder dass die Schlüssel und damit die verschlüsselten Daten durch ungeeignete Ablagen (Zettel auf dem Schreibtisch!) kompromittiert werden. Aber auch die Eingabe (Keylogger, Beobachtung der Eingabe!) und Übertragung der Schlüssel im Netzwerk (verschlüsselt?) könnten zum Sicherheitsproblem werden.

Für das Beispiel in diesem Beitrag wird auf die erste der genannten Möglichkeiten zurückgegriffen: Speicherung der Schlüssel in einer eigenen Tabelle. Die Schlüssel sollen nur für einen bestimmten Zeitraum Gültigkeit haben. Die Tabelle für die Schlüssel wird mit folgendem Befehl angelegt:

```
CREATE TABLE schluesseltabelle
  ( schluessel RAW(300) NOT NULL,
    startdatum DATE NOT NULL,
    enddatum DATE)
```

In diese Tabelle wird mit folgendem Befehl der Schlüssel für den ersten Zeitraum eingefügt:

```
INSERT INTO schluesseltabelle (schluessel, startdatum) VALUES
  ( SYS.DBMS_CRYPTO.RANDOMBYTES(32),
    to_date('AUG-01-2008', 'MON-DD-YYYY'))
```

Bei Bedarf wird durch ein UPDATE auf den erzeugten Datensatz ein Enddatum eingefügt und durch ein INSERT ein neuer Satz mit einem neuen Schlüssel für den nächsten Zeitraum eingefügt.

Das Verschlüsseln der Daten wird über eine eigene Funktion erreicht, die auf das Package DBMS_CRYPTO zurückgreift. Damit ist es leicht möglich, sowohl beim INSERT als auch beim UPDATE zu verschlüsseln – entweder indem

man diese Funktion direkt im entsprechenden SQL-Befehl verwendet oder im Rahmen eines Triggers.

```
CREATE OR REPLACE FUNCTION crypt (eingabe IN VARCHAR2)
RETURN RAW
IS
v_rohdaten         RAW(200);  -- Variable für verschlüsselten Wert
v_schluessel       RAW(32);   -- Variable für 256-bit Schlüssellänge
v_verschluesselung  PLS_INTEGER := -- Kennung des Verschlüsselungsalgorithmus
    SYS.DBMS_CRYPTO.ENCRYPT_AES256 +
    SYS.DBMS_CRYPTO.CHAIN_CBC +
    SYS.DBMS_CRYPTO.PAD_ZERO;
BEGIN
    SELECT schluessel INTO v_schluessel
    FROM schluesseltabelle
    WHERE sysdate BETWEEN startdatum AND nvl(enddatum, sysdate);
    v_rohdaten   := SYS.DBMS_CRYPTO.ENCRYPT(
        src => UTL_I18N.STRING_TO_RAW (eingabe, 'AL32UTF8'),
        typ => v_verschluesselung,
        key => v_schluessel);
RETURN v_rohdaten;
END;
```

Erläuterungsbedürftig ist nur die Verwendung des Parameters SRC: Da DBMS_CRYPTO nur mit den Datentypen LOB, CLOB und RAW arbeitet, müssen Daten vom Typ VARCHAR2 konvertiert

werden. Dies geschieht hier durch Rückgriff auf das Package UTL_I18N, das verlangt, auch den Zielzeichensatz (AL32UTF8) anzugeben.

Das INSERT in eine Tabelle könnte nun also etwa folgendermaßen aussehen:

```
INSERT INTO tabelle (...) VALUES (..., crypt('eintest'), ...)
```

Der lesende Zugriff wird über eine eigene Funktion gesteuert:

```
CREATE OR REPLACE FUNCTION dcrypt (eingabe IN RAW)
RETURN VARCHAR2
IS
v_daten       RAW(200);
v_schluessel  RAW(32);
v_verschluesselung  PLS_INTEGER :=
    SYS.DBMS_CRYPTO.ENCRYPT_AES256 +
    SYS.DBMS_CRYPTO.CHAIN_CBC +
    SYS.DBMS_CRYPTO.PAD_ZERO;
BEGIN
    SELECT schluessel INTO v_schluessel
    FROM schluesseltabelle
    WHERE sysdate BETWEEN startdatum AND nvl(enddatum,
sysdate);
    v_daten    := SYS.DBMS_CRYPTO.DECRYPT(
```

```
        src => eingabe,
        typ => v_verschluesselung,
        key => v_schluessel);
RETURN UTL_I18N.RAW_TO_CHAR(v_daten, 'AL32UTF8');
END;
```

Das SELECT greift auf die verschlüsselte Spalte folgendermaßen zu:

```
SELECT dcrypt(spaltenname)
FROM tabelle
WHERE ...
```

Es empfiehlt sich, den Quellcode der Funktionen zum Beispiel mit der Utility WRAP zu schützen. In der Datei DATEINAME.SQL steht jeweils das Statement, mit dem die Funktion zum Verschlüsseln beziehungsweise Entschlüsseln angelegt wird:

```
wrap iname=dateiname.sql
```

Damit sind die im Quellcode enthaltenen Informationen nicht mehr einfach zugänglich.

3.2 DEKLARATIV MIT ADVANCED SECURITY TRANSPARENT DATA ENCRYPTION (TDE) IN DER ENTERPRISE EDITION

Wenn man sich entschließen kann, die *Advanced Security Option (ASO)* einzusetzen, wird die Verschlüsselung von Benutzerdaten wesentlich einfacher. Damit fallen zwar zusätzliche Lizenzgebühren

an, aber der Erwerb einer Lizenz für die Advanced Security Option erlaubt im Rahmen des Feature *Transparent Data Encryption (TDE)* neben der Verschlüsselung von Benutzerdaten die Verschlüsselung von:

- Backups mit RMAN (nicht zur verschlüsselten Speicherung ohnehin verschlüsselter Daten, sondern zur Verschlüsselung zum Beispiel des Backups der Tablespaces SYSTEM und SYSAUX oder anderer Datenbank-Backups auf dem gleichen Knoten)
- Exports mit DATA PUMP

Hier wird beschrieben, wie man TDE einsetzt, um Benutzerdaten zu verschlüsseln.

3.2.1 EINLEITUNG

TDE wurde in Oracle Database 10g eingeführt. Es ist damit möglich, Tabellenspalten zu verschlüsseln. Allerdings gibt es bei dieser Variante der Verschlüsselung eine ganze Reihe von Einschränkungen. Sie betreffen zum Beispiel die Datentypen, die verschlüsselt werden können, oder auch die Indizes, die auf eine verschlüsselte Spalte gelegt werden dürfen. Ebenso ist es beispielsweise unmöglich, Fremdschlüsselspalten zu verschlüsseln.

Seit Oracle Database 11g kann man nun außerdem ganze Tablespaces verschlüsseln. Dabei gibt es keinerlei Einschränkungen bezüglich Datentypen, Indizes und so weiter. Weil das Verschlüsseln und Entschlüsseln von Tablespaces im Rahmen der Ein-/Ausgabeoperation erfolgt und zusätzlich die neuen Ver- und Entschlüsselungsmöglichkeiten neuerer CPUs nutzt, ist es auch noch performanter als das Verschlüsseln vieler einzelner Spalten. Es ist deshalb sicherlich verständlich, dass der Tablespace-Verschlüsselung im Allgemeinen der Vorzug gegeben wird.

3.2.2 VORBEREITUNG: WALLETS, KEYSTORES, PASSWÖRTER UND SCHLÜSSEL

Es mag vielleicht auf den ersten Blick verwundern, aber die größte Aufmerksamkeit sollte im Rahmen von TDE dem Umgang mit sogenannten Wallets, Keystores, Passwörtern und Schlüsseln gewidmet werden. Das liegt daran, dass das eigentliche Verschlüsseln sehr einfach deklarativ beim Anlegen oder Ändern der zu verschlüsselnden Objekte veranlasst wird. Dagegen erfordert der Umgang mit Wallets, Keystores, Passwörtern und Schlüsseln schon einige Vorüberlegungen.

Verschlüsselungsverfahren wie AES oder Blowfish sind jedermann zugänglich. Es ist kein Geheimnis, wie sie funktionieren. Allerdings verwenden diese Verfahren einen

vom Anwender zu ergänzenden Wert, der das Ergebnis der Verschlüsselung bestimmt: den sogenannten Schlüssel. Wer den Schlüssel kennt und weiß, welches Verfahren zur Verschlüsselung angewendet wurde, hat Zugriff auf die verschlüsselten Informationen. Da die Bestimmung des Verschlüsselungsverfahrens für Experten relativ einfach ist, liegt die Sicherheit verschlüsselter Informationen also im Schlüssel. Darum ist auch die Sicherung (Verwaltung) und ausreichende Komplexität (Erzeugung) der Schlüssel absolut entscheidend.

Verwaltung und Erzeugung der Schlüssel erfolgen im Rahmen von TDE immer komplett durch die Datenbank. Dabei wird nicht zwischen der Spalten- und Tablespace-Verschlüsselung unterschieden. Das Verfahren ist zweistufig: Jede Tabelle und jedes Tablespace verfügt über einen eigenen Schlüssel, mit dem die dazugehörigen Daten ver- und entschlüsselt werden. Dieser Schlüssel wird nach den Vorgaben des Anwenders von Oracle erzeugt und innerhalb der Tabelle oder des Tablespace gespeichert. Die zweite Stufe ist der sogenannte *Master Key*, ein Schlüssel, der ausschließlich dazu dient, die Schlüssel der Tabellen und Tablespaces zu ver- und entschlüsseln. Auch dieser Schlüssel wird durch die Datenbank automatisch erzeugt, allerdings ohne irgendeine Einflussnahme des Anwenders. Außerdem wird dieser Master Key außerhalb der Datenbank gespeichert.

Zur Speicherung des Master Keys wird entweder eine verschlüsselte Datei – ein sogenanntes *Wallet* oder *Keystore* – auf der Betriebssystemebene genutzt oder ab Oracle Database 11g bei sehr hohen Sicherheitsanforderungen ein eigenes Stück Hardware, ein sogenanntes *Hardware Security Modul (HSM)*. In der Datei SQLNET.ORA wird festgelegt, welche Variante genutzt wird.

Die folgenden Aktionen sind auch über die grafische Schnittstelle des *Enterprise Manager Grid* oder *Cloud Control* durchführbar, werden aber hier über Befehle, die auf der Kommandozeile eingegeben werden können, dargestellt.

Für dieses Dojo soll der Master Key zunächst in einem Wallet/Keystore angelegt und verwaltet werden. Der Umgang mit einem HSM wird später im Zusammenhang mit Informationen zu dem Produkt Oracle Key Vault (OKV) vorgestellt.

In der Datei SQLNET.ORA wird also, wie oben bereits erwähnt, zunächst angegeben, dass und in welchem Verzeichnis dieses Wallet anzulegen oder zu finden ist. Obwohl es für unterschiedliche Betriebssysteme unterschiedliche Default-Einstellungen für den Speicherort des Wallets gibt, ist es dennoch empfehlenswert, den Speicherort explizit zu benennen:

```
ENCRYPTION_WALLET_LOCATION =  (SOURCE = (METHOD = FILE)
                              (METHOD_DATA = (DIRECTORY =
                               $ORACLE_HOME/network/admin)))
```

Der Eintrag hat zunächst keinerlei Konsequenzen. Erst der nächste Schritt vollendet den Arbeitsschritt. In der Version 11 der Datenbank oder in einer Nicht-Container-Datenbank der Version 12 besteht er aus einem einzigen Befehl:

```
ALTER SYSTEM SET ENCRYPTION KEY IDENTIFIED BY “dojopasswort”
```

In einer Container-Datenbank sind drei Befehle nötig, um den gleichen Arbeitsschritt durchzuführen:

```
ADMINISTER KEY MANAGEMENT
CREATE KEYSTORE  ‘/etc/wallet/orcl‘ IDENTIFIED BY dojopasswort

ADMINISTER KEY MANAGEMENT
SET KEYSTORE OPEN IDENTIFIED BY dojopasswort

ADMINISTER KEY MANAGEMENT
SET KEY IDENTIFIED BY dojopasswort WITH BACKUP
```

Nach diesem Befehl oder nach diesen Befehlen hat das Datenbanksystem in dem in der SQLNET.ORA angegebenen Verzeichnis eine verschlüsselte Datei mit dem Namen EWALLET.P12 angelegt und in dieser Datei den Master Key gespeichert, der *nicht* identisch ist mit dem Passwort „dojopasswort“. Wird das Passwort im Rahmen des Befehls

ALTER SYSTEM ohne Anführungszeichen eingegeben, muss es später in Großbuchstaben eingegeben werden. Die Groß- und Kleinschreibung des Passwortes wird bei der Verwendung des Befehls ADMINISTER KEY MANAGEMENT auch ohne Hochkommata berücksichtigt.

Eine Änderung des Master Keys ist über den gleichen Befehl ALTER SYSTEM beziehungsweise ADMINISTER KEY MANAGEMENT SET KEY möglich (mit gültigem Passwort!). Der neue Master Key wird ebenfalls in der Datei EWALLET.P12 gespeichert.

Nach den Erläuterungen oben ist sicherlich klar, dass in administrativer Hinsicht das Wallet/Keystore das zentrale Element von TDE darstellt. Deshalb sind einige Hinweise dazu unbedingt angebracht.

Zugriff auf verschlüsselte Daten erhält man nur über ein geöffnetes Wallet/Keystore. Das Öffnen erfolgt bei jedem Start der Datenbank je nach Datenbankversion über den Befehl:

```
ALTER SYSTEM SET ENCRYPTION WALLET OPEN IDENTIFIED BY
"dojopasswort"
```

oder über

```
ADMINISTER KEY MANAGEMENT
SET KEYSTORE OPEN IDENTIFIED BY dojopasswort
```

Falls das Passwort zwischenzeitlich verändert wurde, muss natürlich das letzte gültige Passwort angegeben werden. Sofern die Sensitivität der Daten es erfordert, kann im Rahmen der Aufgabenverteilung und Funktionstrennung die Eingabe des Befehls einem eigenen DBA übergeben werden, den man auch als Security-DBA bezeichnet. Nur dieser würde das zum Öffnen des Wallets/Keystores nötige Passwort kennen.

Nach dem Öffnen des Wallets/Keystores wird – anders als bei der Verwendung eines HSMs – der Master Key ausgelesen und in der SGA leicht verschlüsselt vorgehalten. Das Wallet/Keystore selbst wird deshalb von der laufenden Datenbank nicht mehr benötigt. Das eröffnet auch die Möglichkeit, das Wallet/Keystore zum Beispiel auf einem Stick zu speichern und diesen Stick, nach der Öffnung der Datenbank, vom Rechner abzuziehen und an einer sicheren Stelle aufzubewahren.

Es ist möglich und vorwiegend für Test- und Entwicklungsumgebungen sinnvoll, das Wallet/Keystore automatisch bei jedem Start der Datenbank zu öffnen. Man spricht dann von einem sogenannten *auto open* oder *auto login wallet/keystore*. Wenn man diese Möglichkeit in produktiven Umgebungen verwendet, sollte man das Sicherheitsrisiko, das durch den Diebstahl solcher Wallets/Keystores bei gleichzeitigem Diebstahl der Datendateien entsteht, dadurch reduzieren,

dass man diese Wallets/Keystores als sogenannte *local auto open* oder *local auto login wallets/keystores* anlegt. Denn diese lassen sich dann nur auf dem Rechner öffnen, auf dem sie auch angelegt wurden.

Das automatische Öffnen des Wallets in Datenbanken der Version 10 und 11 ist mit dem Oracle Wallet Manager (OWM) zu veranlassen. Es wird dann im selben Verzeichnis, in dem auch die Datei EWALLET.P12 liegt, eine Datei EWALLET.SSO angelegt, in der das benötigte Passwort verschlüsselt gespeichert ist. Mit dem OWM ist auch das Passwort für das Wallet zu ändern, das – um nochmals darauf hinzuweisen – *nicht* identisch ist mit dem Master Key.

Der Master Key kann mit dem Befehl

```
ALTER SYSTEM SET ENCRYPTION WALLET CLOSE
-- OHNE identified by "dojopasswort"!
```

aus der SGA entfernt werden. Um einen Zugriff auf verschlüsselte Daten wieder zu ermöglichen, muss das Wallet erneut explizit geöffnet werden.

In einer Container-Datenbank wird das automatische Öffnen des Keystores veranlasst über den Befehl:

```
ADMINISTER KEY MANAGEMENT
CREATE (LOCAL) AUTO_LOGIN KEYSTORE
```

```
FROM KEYSTORE '/etc/wallet/orcl' IDENTIFIED BY dojopasswort
```

Das Passwort wird mit folgendem Befehl geändert:

```
ADMINISTER KEY MANAGEMENT
ALTER KEYSTORE PASSWORD
IDENTIFIED BY dojopasswort SET neuesdojopasswort WITH BACKUP
```

Wallets/Keystores sollten nach jeder Änderung des Passworts oder des Master Keys gesichert werden. Es gibt *keine* Möglichkeit (kein Hintertürchen oder *backdoor*), verschlüsselte Daten ohne den Master Key zu entschlüsseln. Geht das Wallet/Keystore verloren, sind die Daten je nach Szenario unwiderruflich verloren.

Auch im Rahmen des normalen Backups der Datenbank sollte man sich eine Strategie für das Sichern des Wallets/Keystores überlegen. In der Version 12 der Datenbank erfolgt das automatisch durch die Verwendung der Klausel WITH BACKUP bei den entsprechenden Befehlen. In den Versionen 10 und 11 muss der verantwortliche Mitarbeiter selbst für dieses Backup sorgen. Vor allem bei Verwendung eines *auto login wallet/keystore* wird dringend davon abgeraten, das Wallet/Keystore zusammen mit dem Backup abzulegen: Der Diebstahl des Backups würde dazu führen, dass der Dieb ganz einfach Zugriff auf die verschlüsselten Daten erlangen kann.

3.2.3 TABELLENSPALTEN, SECUREFILE LOBS UND TABLESPACES VERSCHLÜSSELN

Nach dem Anlegen des Wallets/Keystores und des Master Keys können Tabellenspalten, SecureFile LOBs und auch ganze Tablespaces verschlüsselt werden – ganz einfach, indem beim Anlegen der Objekte die Verschlüsselungsklausel genutzt wird:

```
CREATE TABLE tabellenname
     (spaltenname varchar2(2000) ENCRYPT USING 'AES256' SALT,
     lobspalte blob)
     LOB (lobspalte) STORE AS SECUREFILE(ENCRYPT USING
     'AES256')

CREATE TABLESPACE sicheristsicher
     DATAFILE '/app/oracle/oradata/dojo.dbf' SIZE 10G
     ENCRYPTION USING 'AES256' DEFAULT STORAGE (ENCRYPT)
```

Alle Daten, die in die Tabelle oder in das Tablespace eingefügt oder dort manipuliert werden, werden unter Beibehaltung der bekannten Befehle ohne weitere Klauseln – in der Oracle-Terminologie „transparent" – ver- beziehungsweise entschlüsselt. Neben dem in den Beispielen genannten zur Zeit stärksten in der Datenbank verfügbaren Verschlüsselungsalgorithmus AES256 sind weitere Algorithmen verfügbar. Länderspezifische Algorithmen oder Open-Source-Verfahren wie Blowfish können nicht verwendet werden.

Innerhalb einer Tabelle muss für alle verschlüsselten Spalten der gleiche Algorithmus zur Verschlüsselung gewählt werden.

Der Benutzer kann bei der Spaltenverschlüsselung zwischen SALT und NO SALT wählen. Soll ein Index auf die Spalte gelegt werden, ist nur NO SALT möglich. Die Option SALT veranlasst, dass dem zu verschlüsselnden Wert zusätzliche Zeichen hinzugefügt werden. Dadurch ergibt die Verschlüsselung gleicher Werte bei gleichem Verschlüsselungsverfahren und gleichem Schlüssel jeweils ein anderes Verschlüsselungsergebnis. Das erhöht zwar die Sicherheit, lässt allerdings auch die Datenmenge pro verschlüsseltem Wert um 16 Bytes anwachsen. Sofern die Datenmenge ein Problem darstellt, kann bei der Spaltenverschlüsselung auch darüber nachgedacht werden, das standardmäßig aktivierte Prüfsummenverfahren SHA-1 zu deaktivieren. Dieses Deaktivieren erfolgt bei der Spaltendeklaration über einen Parameter mit Namen NOMAC. Ein Verzicht auf die Prüfung auf Manipulation der verschlüsselten Daten reduziert die Datenmenge noch einmal um 20 Bytes pro verschlüsseltem Wert.

Während Tabellenspalten mit ALTER TABLE auch nachträglich verschlüsselt werden können und die Verschlüsselung nachträglich auch komplett aufgehoben werden kann,

ist eine nachträgliche Verschlüsselung oder Aufhebung der Verschlüsselung eines Tablespace nicht möglich. Für beides – nachträgliche Verschlüsselung und Aufhebung der Verschlüsselung – muss ein Tablespace als verschlüsselt oder unverschlüsselt *neu* angelegt werden. Vorhandene Daten müssten dann mit einem CREATE TABLE AS SELECT, mit einem Export/Import oder anderen Verfahren in das neue Tablespace verschoben werden. Eine nachträgliche Schlüsseländerung des Master Keys, die zum Beispiel nach dem Pseudostandard PCI-DSS der Kreditkartenbranche in regelmäßigen Abständen durchgeführt werden muss, ist – wie bereits dargestellt – möglich.

Eine Schlüsseländerung für Tabellenspalten kann zudem auf der Ebene der Tabelle durchgeführt werden. Während bei der Änderung des Master Keys lediglich die Tabellenschlüssel neu verschlüsselt werden, was natürlich wenig zeitaufwendig ist, führt die Neuverschlüsselung des Tabelleninhalts zu einem UPDATE auf alle Zeilen, die in einer verschlüsselten Spalte einen Wert enthalten. Je nach Größe der Tabelle kann das extrem viel Zeit beanspruchen und zusätzlich Endanwender behindern, die im Rahmen ihrer Aufgaben Zeilen ändern müssen.

3.3 TDE UND DATENBANKEN DER VERSION 12

Es wurde ja bereits darauf hingewiesen, dass Container-Datenbanken andere SQL-Befehle verwenden als Nicht-Container-Datenbanken oder Datenbanken der Versionen 10 und 11. Aber es gibt ein paar weitere wichtige Details, die erwähnt werden sollen.

So ist es vielleicht nicht selbstverständlich, dass es pro Container-Datenbank immer nur einen Keystore gibt. In diesem Keystore speichern alle Pluggable Databases ihre Master Keys. Diese Keys können exportiert und importiert werden, damit der Transport einer Pluggable Database mit verschlüsselten Daten möglich ist. Der Keystore einer Container-Datenbank kann auch komplett mit dem Keystore einer anderen Container-Datenbank zusammengeführt werden.

Außerdem ist bemerkenswert, dass ein Keystore immer erst auf der Ebene der Container-Datenbank geöffnet werden muss, bevor einzelne Pluggable Databases es für sich öffnen und nutzen können.

Erweiterungen gibt es auch im Bereich der Trennung von Funktionen und Aufgaben. Während in älteren Datenbankversionen häufig als SYSDBA gearbeitet wurde, wenn es um TDE ging, erlaubt die Version 12 das Festlegen eines eigenen Systemprivilegs sowie im Rahmen der Installation einer eigenen Betriebssystemrolle. Die Rolle kann selbst benannt

werden, das Privileg heißt SYSKM. Damit kann die Person, der das Privileg und/oder die Rolle zugewiesen wird, den Befehl ADMINISTER KEY MANAGEMENT nutzen sowie auf die relevanten *Data Dictionary Views* zugreifen.

Hinweis: Über das Privileg kann bei geöffneter Datenbank gearbeitet werden, während die Betriebssystemrolle auch bei noch nicht geöffneter Datenbank zur Verfügung steht. Letzteres ist beim Starten der Datenbank wichtig, falls der Keystore explizit geöffnet werden muss.

4 Noch mehr Sicherheit für Schlüssel, Wallets und Keystores

Im Abschnitt 3.2.2 wurde bereits betont, dass der Umgang mit Wallets/Keystores beim Einsatz von TDE größte Aufmerksamkeit erfordert. Das Produkt *Oracle Key Vault (OKV)*, das im August 2014 zum Einsatz freigegeben wurde, unterstützt Sicherheitsverantwortliche dabei und ist somit die perfekte Ergänzung zu TDE.

Oracle Key Vault (OKV) bietet zwei Funktionen:

- Das Produkt kann einerseits Wallets/Keystores ersetzen, indem es als sogenanntes *Hardware Security Modul (HSM)* verwendet wird. Diese Möglichkeit stand beim Einsatz

von TDE schon seit der Datenbankversion 11.2 grundsätzlich zur Verfügung. Da Oracle selbst kein eigenes HSM-Produkt anbieten konnte, haben Unternehmenskunden dann auf Produkte anderer Anbieter zurückgegriffen. Das hat sich mit OKV geändert.

- Andererseits kann Oracle Key Vault viele Formen von Wallets und Keystores komplett speichern und damit als Backup Repository für diese Wallets und Keystores dienen.

Abhängig vom Bedrohungsszenario kann die Entscheidung gegen den Einsatz von Wallets/Keystores und für den Einsatz eines HSMs durchaus sinnvoll sein, denn:

- ein HSM bietet mehr Sicherheit: Eine Betriebssystemdatei kann leichter gestohlen (kopiert) werden, als ein HSM, das in der Regel als speziell gesicherte Steckkarte in einem Rechner eingebaut ist oder als eigenes Gerät geschützt in einem Rechenzentrum steht.
- ein HSM kann anders als ein Wallet/Keystore systemübergreifend verwendet werden. Das erlaubt eine gemeinsame Nutzung von Schlüsseln – was wiederum zum Beispiel den Einsatz von TDE auf RAC-Installationen perfekt unterstützt.
- ein HSM kann von mehreren Anwendungen genutzt werden. Das erleichtert das Konsolidieren und Verwalten von Passwörtern und Schlüsseln.

Ein Repository für Wallets kann dagegen sinnvoll sein:

- wenn man befürchten muss, dass lokale Wallets gelöscht werden und sogar die eigenen Sicherungen dieser Wallets verloren sind oder nicht einmal gemacht wurden; da auch Oracle hier nicht weiterhelfen kann, wären dann je nach Szenario alle verschlüsselten Daten komplett verloren.
- wenn man vordringlichst die Verfügbarkeit der Datenbank im Blick hat: Ohne einen Zugriff auf OKV/HSM würde die Datenbank nicht neu gestartet werden können. Das Szenario ist ausgesprochen unwahrscheinlich, denn wegen seiner Bedeutung kann OKV oder ein HSM auf Hochverfügbarkeit ausgelegt werden; aber weil der eine oder andere Sicherheitsbeauftragte vielleicht denkt „anything that can go wrong, will go wrong“ mag diese Person sich mit ständig lokal verfügbaren Wallets/Keystores wohler fühlen.

4.1 INSTALLATION DES OKV

Die Installation von OKV ist denkbar einfach, denn das Produkt wird in Form eines ISO-Images als *Software Appliance* geliefert. Das Konzept Software Appliance ist bereits von *Oracle Audit Vault and Database Firewall* bekannt: Der Kunde stellt einen X86-64bit-Rechner zur ausschließlichen

Nutzung für OKV zur Verfügung. Auf dem Rechner wird die OKV-Software durch das Starten des ISO-Images installiert. Der gesamte installierte Softwarestack besteht aus dem Betriebssystem *Oracle Enterprise Linux (OEL)*, einer Oracle-Datenbank und einer grafischen Benutzeroberfläche, der *Oracle Key Vault Management Console*. Nach der Installation dürfen diese Komponenten in keiner Weise verändert, sondern nur in festgelegtem Ausmaß konfiguriert werden.

Der Installationsprozess erwartet nur zwei Eingaben: Angaben zur IP-Adresse (IP-Adresse, Mask und sofern nötig Gateway) für den OKV und ein Einmalpasswort, das OKV als Passphrase bezeichnet. Es ermöglicht das erstmalige Einloggen in die OKV Management Console.

4.2 KONFIGURATION DES OKV

Nach Abschluss der eigentlichen Installation müssen noch einige Benutzer und ihre Passwörter eingerichtet werden. Außerdem muss ein Passwort für Notfälle festgelegt werden. Als Notfall ist in diesem Zusammenhang zum Beispiel ein eventuell erforderliches Recovery des OKV zu verstehen.

Das Einrichten der Benutzer und des Notfallpasswortes erfolgt in der Console. Sie ist über einen Browser unter der bei der Installation angegebenen URL erreichbar. In dieser ersten Version des OKV macht man sich das Leben leichter,

wenn man den Browser und die Tastaturbelegung auf die englische Sprache einstellt.

Nach dem ersten Einloggen in die OKV Management Console werden die folgenden Benutzer angelegt:

- ein Key-Administrator, der zum Beispiel den Zugriff auf Wallets und Schlüssel kontrolliert. Er kann auch weitere Key-Administratoren einrichten.
- ein Administrator, der für die Verwaltung des OKV zuständig ist, zum Beispiel für das Starten und Stoppen des OKV, für Backups und Recovery, Einrichten von Hochverfügbarkeit sowie von Endpoints und Benutzern. Er kann auch weitere Administratoren einrichten.
- ein Audit-Manager, der den Audit-Trail des OKV verwaltet und weitere Audit-Manager einrichten kann.
- ein Root-User, der sich bei Bedarf, zum Beispiel zum Ausführen festgelegter Skripte oder im Recoveryfall als Benutzer *root* auf dem OKV einloggen kann.
- ein Support-User, der sich mit SSH auf dem System anmelden kann.

Die Aufgaben, die die ersten drei Benutzer wahrnehmen, können auch von einem Benutzer oder auch von zwei Benutzern wahrgenommen werden. Zum Abschluss wird das Passwort für

Notfälle angegeben. Außerdem wird man für Produktivumgebungen einstellen, wie die Zeitsteuerung des OKV erfolgen soll und welche Name-Server zur Verfügung stehen.

4.2.1 ENDPOINT EINRICHTEN

Systeme, die Oracle Key Vault (OKV) nutzen, werden *Endpoints* genannt. Endpoints werden in Zusammenarbeit zwischen einem OKV-System-Administrator und einem Endpoint-Administrator eingerichtet. In der Terminologie von OKV heißt das „Administrator-initiierte Einrichtung“ (administrator-initiated enrollment). Dabei vergibt der OKV-System-Administrator einen Namen für den neuen Endpoint, legt Typ und Betriebssystem des Endpoints fest und erzeugt schließlich ein sogenanntes *Token*, das der Endpoint-Administrator für den Start der Konfiguration übermittelt erhält. Ein Token ist nur einmal einsetzbar.

4.3 DATENBANK EINRICHTEN

Ein Benutzer, der die Berechtigung hat, Dateien als Eigentümer der Oracle-Datenbanksoftware auf dem Datenbankserver zu installieren, ruft im Browser die OKV Management Console auf. Er versucht nicht, sich einzuloggen, sondern klickt am Fuß der Seite auf den Link „Endpoint Enrollment and Software Download“.

Auf der Seite, die sich öffnet, wird das Token eingegeben. Außerdem werden die Angaben zum Typ und Betriebssystem gemacht – hier Database und Linux. Nach dem Anklicken von Submit Token und Enroll wird eine Datei namens OKVCLIENT.JAR zum Speichern angeboten. Sie kann in einem beliebigen Verzeichnis abgelegt werden.

Die Jar-Datei wird ausgepackt. Dazu muss mindestens JDK 1.5 installiert sein. Außerdem ist die Variable JAVA_HOME zu setzen. Auch der Pfad zum *Java Executable* sollte explizit gesetzt sein. Auf die Aufforderung, ein Passwort einzugeben und zu bestätigen, wird für das Beispiel dieses Artikels „DojoSec.1“ eingegeben. Die Eingabe erfolgt verdeckt, aber das Passwort wird beim Einrichten von TDE noch gebraucht und deshalb hier offengelegt.

```
-- Falls der Befehl abgetippt wird, den Zeilenumbruch nicht übernehmen
[oracle@hwf Desktop]$ java -jar okvclient.jar -d /oracle/app/
oracle/product/okvutil -v
Detected JAVA_HOME: /usr/lib/jvm/java-1.6.0-openjdk-
1.6.0.0.x86_64/jre
Enter new Key Vault endpoint password ( for auto-login):
Confirm new Key Vault endpoint password:
Oracle Key Vault endpoint software installed successfully.
```

Das weitere Vorgehen unterscheidet sich danach, ob OKV als HSM oder als Wallet/Keystore Repository verwendet werden soll.

4.4 VERWENDUNG VON OKV ALS REPOSITORY

OKV kann zur Zeit als Backup Repository für folgende Dateitypen verwendet werden:

- Oracle Wallets und Keystores
- Java Keystores (JKS)
- Java Cryptography Extension Keystores (JCEKS)
- SSH Key Files
- Kerberos Keytabs

Dabei werden Dateien wie Kerberos Keytabs als LOBs gespeichert, während die Inhalte der aus dem Oracle-Umfeld bekannten Wallets/Keystores ausgelesen und in sogenannten virtuellen Wallets abgelegt werden. Sowohl beim Einsatz als HSM als auch auch beim Einsatz als Repository muss beachtet werden, dass OKV zur Zeit nur mit Systemen zusammenarbeitet, auf denen das Betriebssystemen Linux x86-64 (Versionen 5 und 6) sowie Solaris (Versionen 10 und 11) läuft.

4.4.1 VORBEREITUNG IN OKV

Das Erstellen eines Wallet Repository erfordert zunächst das Anlegen sogenannter *virtual wallets* in OKV. Der OKV-Systemadministrator legt es in diesem Beispiel mit

dem Namen DOJOWALLET an. Diesen Namen erhält der Endpoint-Administrator ebenfalls übermittelt. Außerdem muss der OKV-Systemadministrator eine Verbindung zwischen Endpoint und virtual wallet herstellen sowie über die Console Angaben zur Zugriffsberechtigung machen. Da für dieses Dojo der Endpoint-Administrator kompletten Zugriff erhalten soll, werden ihm Lesen, Ändern und Verwalten erlaubt.

4.4.2 DATENBANKSERVER EINRICHTEN

Es wird lediglich das durch das Entpacken der Datei OKVCLIENT.JAR angelegte Werkzeug OKVUTIL benötigt. Die Vorbereitungen der Datenbank sind also bereits abgeschlossen.

4.4.3 TDE WALLET IN OKV SICHERN

Das Sichern eines Wallets in OKV geschieht über einen Aufruf des Hilfsprogramms OKVUTIL:

```
-- Falls der Befehl abgetippt wird, den Zeilenumbruch nicht übernehmen
orcl /home/oracle/okvutil/bin> ./okvutil upload -l  '$ORACLE_
HOME/network/admin' -t wallet -g "DOJOWALLET"
Enter source wallet password:
Enter Oracle Key Vault endpoint password:
Upload succeeded
```

Dabei steht nach dem Parameter „-l“ das Verzeichnis, in dem das Wallet EWALLET.P12 gespeichert ist, nach „-t“ der Typ der Datei und nach „-g“ der Name des *virtual wallet* (Groß- und Kleinschreibung ist zu beachten!), in dem die Informationen aus EWALLET.P12 abgelegt werden sollen. Außerdem werden das Wallet-Passwort und auch das Passwort für den Endpoint abgefragt. Wie oben angegeben lautet es „DojoSec.1“.

Um gleich auch zu testen, dass der umgekehrte Weg – nämlich das Herunterladen eines Wallets – funktioniert, wird das lokale Wallet gelöscht und aus OKV neu heruntergeladen:

```
orcl /oracle/app/oracle/product/11.2.0/dbhome_1/network/
admin> ll *wall*
-rw-r--r-- 1 oracle oinstall 2845 Sep 30 16:14 ewallet.p12
orcl /oracle/app/oracle/product/11.2.0/dbhome_1/network/
admin> rm *wall*
orcl /oracle/app/oracle/product/11.2.0/dbhome_1/network/
admin> ll *wall*
ls: *wall*: No such file or directory
orcl /oracle/app/oracle/product/11.2.0/dbhome_1/network/
admin> cd
    /home/oracle/okvutil/bin
orcl /home/oracle/okvutil/bin> ./okvutil download -l
“$ORACLE_HOME/network/admin” -t wallet -g “DOJOWALLET”
```

```
Enter new wallet password (RETURN for auto-login):
Enter Oracle Key Vault endpoint password:
Download succeeded
orcl /home/oracle/okvutil/bin> cd $ORACLE_HOME/network/admin
orcl /oracle/app/oracle/product/11.2.0/dbhome_1/network/
admin> ll *wall*
-rw------- 1 oracle oinstall 5733 Sep 30 16:17 cwallet.sso
-rw-r----- 1 oracle oinstall 5656 Sep 30 16:17 ewallet.p12
```

Die gesamte Aktion ist abgelaufen wie erwartet: Das neue Wallet existiert an der angegebenen Stelle. Obwohl der Sourcecode ansonsten nicht näher erläutert werden muss, ist vielleicht doch ein Hinweis angebracht: Die Abfrage nach dem neuen Passwort für das Wallet ist nur mit RETURN bestätigt worden. Deshalb wurde automatisch ein *auto open wallet* angelegt. Das ist auch an der Existenz der Datei CWALLET.SSO erkennbar.

4.5 OKV ALS HSM NUTZEN

4.5.1 DATENBANKSERVER EINRICHTEN

Wie oben bereits erwähnt, ist es möglich, Master Keys für Transparent Data Encryption (TDE) direkt in OKV zu speichern, OKV also als HSM zu nutzen. Dazu wird mindestens eine Oracle-Datenbank der Version 11.2 benötigt.

Ausgangsbasis ist für dieses Dojo eine Datenbank, die noch nicht für TDE konfiguriert wurde. Falls eine solche Konfiguration bereits stattgefunden hat und nun statt mit Wallets/Keystores direkt mit OKV gearbeitet werden soll, besteht aber durchaus eine einfache Migrationsmöglichkeit. Das Administrationshandbuch beschreibt die Details.

Für den Einsatz als HSM muss zusätzlich eine Library verfügbar sein, die den Zugriff auf das PKCS#11-kompatible OKV ermöglicht. Diese Library ist ebenfalls Teil der Datei OKVCLIENT.JAR. Da das System die Library in einem bestimmten Verzeichnis erwartet, stellt Oracle ein Skript namens *root.sh* bereit, das das Verzeichnis anlegt und die Library dort ablegt. Die Library heißt übrigens *liborapkcs.so*.

Als Benutzer *root* wird also das Shell-Skript *root.sh* aus dem Verzeichnis BIN von OKVUTIL ausgeführt.

```
[root@hwf bin]# pwd
/oracle/app/oracle/product/okvutil/bin
[root@hwf bin]# ./root.sh
Creating directory: /opt/oracle/extapi/64/hsm/oracle/1.0.0/
Copying PKCS library to /opt/oracle/extapi/64/hsm/oracle/1.0.0/
Setting PKCS library file permissions
Installation successful.
```

Damit ist die Datenbank für das Arbeiten mit OKV als HSM vorbereitet.

4.5.2 TDE EINRICHTEN

Das Einrichten von TDE erfolgt in der bekannten Art und Weise. Zunächst wird die Datei SQLNET.ORA um den Parameter ENCRYPTION_WALLET_LOCATION ergänzt:

```
ENCRYPTION_WALLET_LOCATION=(SOURCE=(METHOD=HSM))
```

Eine Abfrage auf V$ENCRYPTION_WALLET zeigt, dass ein HSM als Speicherort für Schlüssel erwartet wird:

```
SQL> SELECT * FROM v$encryption_wallet;
WRL_TYPE        WRL_PARAMETER                          STATUS
--------------  -------------------------------------  -------
HSM                                                    CLOSED
```

Das bekannte Statement ALTER SYSTEM erzeugt nun den ersten Master Key. Das Passwort, das diesem Befehl übergeben wird, ist das, welches oben beim Entpacken von OKVCLIENT.JAR festgelegt wurde, nämlich „DojoSec.1“. Dass das Arbeiten mit dem OKV auch funktioniert, belegt das Anlegen eines verschlüsselten Tablespace:

```
SQL orcl> ALTER SYSTEM set encryption key IDENTIFIED BY “DojoSec.1”;
System altered.
SQL orcl> CREATE TABLESPACE sicheristsicher
    2   DATAFILE ‘/home/oracle/dateiname.dbf’ SIZE 100G
    3   ENCRYPTION USING ‘AES256’ DEFAULT STORAGE (ENCRYPT);

Tablespace created.
```

```
SQL orcl> SELECT tablespace_name, encrypted FROM dba_
tablespaces
    2   WHERE tablespace_name like 'SICHER%';

TABLESPACE_NAME                 ENCRYPTED
------------------------------  ---------
SICHERISTSICHER                 YES
```

4.6 BERICHTE UND ALERTS

Neben den rein funktionalen Möglichkeiten des Arbeitens mit OKV, die im Rahmen eines Dojos zum Thema *Verschlüsselung* in erster Linie interessant sind, soll abschließend noch darauf hingewiesen werden, dass OKV auch darüber hinaus noch einige interessante Features bietet:

- Bestandteil von OKV ist ein eigenes Auditing, das jeden Zugriff auf die dort gespeicherten Objekte nachvollziehbar macht. Dabei muss die Auswertung nicht selbst vorgenommen werden, sondern ist über vorgefertigte Berichte zugänglich.
- Zu jedem Objekt, das in OKV gespeichert ist, können Informationen zum Einsatzgebiet abgefragt werden. So wird zum Beispiel erkennbar, welche Schlüssel für TDE auf welchen Endpoints eingesetzt werden, ob es sich um ein *private* oder *public keys* oder um Zertifikate und so weiter handelt.

- Alerts können die verantwortlichen Anwender darauf aufmerksam machen, dass zum Beispiel der Zeitpunkt gekommen ist, einen Master Key zu ändern oder ein Backup der OKV-Informationen anzufertigen (für dieses Backup stehen ebenfalls vorkonfigurierte Prozeduren zur Verfügung).

5 Fazit

In diesem Dojo sollte klar werden, dass jede Datenbank durch Verschlüsselung sicherer gemacht werden kann. Neben dem häufig kaum erwähnenswerten Aufwand kommt hinzu, dass die früher häufig geäußerte Befürchtung, sich durch den Einsatz von Verschlüsselungstechniken die Performance zu verschlechtern, heute nur noch in extremen Ausnahmefällen gilt. Im Gegenteil bestätigen Kunden immer wieder, dass sie nach der Einführung von Netzwerk- und/oder Datenverschlüsselung keinerlei Performanceunterschiede für ihre Anwendungen erkennen können.

6 Weitere Informationen

Natürlich ist dieses Dojo kein Ersatz für die Lektüre der relevanten Kapitel in den Handbüchern. Im Einzelnen sind hier zu nennen:

- *Oracle Database 2 Day + Security Guide:*
  Kurzeinführung zu TDE
- *Oracle Database Security Guide:*
  gründlicher Überblick über Netzwerkverschlüsselung, prozedurale und deklarative Datenverschlüsselung (TDE)
- *Oracle Database Advanced Security Guide:*
  umfassende Darstellung von TDE
- *Oracle Database Net Services Administrator's Guide und Oracle Database Net Services Reference:*
  komplette Informationen für den Umgang mit der Netzwerkkonfiguration bei TDE
- *Oracle Database PL/SQL Packages and Types:*
  detaillierte Informationen zu DBMS_CRYPTO für die prozedurale Verschlüsselung
- *Oracle Key Vault Administrator's Guide:*
  vollständige Informationen zum Einsatz von OKV

Außerdem bietet auch *My Oracle Support* (MOS) eine ganze Reihe von Informationen zu allen Themen, zum Beispiel:

- Dokument 863071.1
  „Several Examples of Using DBMS_CRYPTO to Encrypt/Decrypt Table Data“
- Dokument 1228046.1
  „Master Note For Transparent Data Encryption (TDE)“
- Dokument 736510.1
  „Step by Step Guide To Configure SSL Authentication“
- Dokument 264080.1
  „An Introduction to PKI and SSL“
- Dokument 166492.1
  „Oracle Advanced Security SSL Troubleshooting Guide“

Aktuelle Informationen zum Thema Datenbank-Security sind auch immer auf den Seiten der Datenbank-Community unter *https://blogs.oracle.com/dbacommunity_deutsch* zu finden.

NOTIZEN

Zugriff auf die komplette
Oracle Dojo-Bibliothek unter
http://tinyurl.com/dojoonline

ORACLE®

Herausgeber: Günther Stürner, Oracle Deutschland B.V.
Design: volkerstegmaier.de // Druck: Stober GmbH, Eggenstein

ORACLE®