Verschlüsseln, auf jeden Fall **verschlüsseln!**

Heinz-Wilhelm Fabry, ORACLE Deutschland B.V. & Co. KG

Verschlüsseln von Datenübertragung und von Daten auf Speichermedien ist trotz aller gegenwärtigen Diskussionen um die grundsätzliche Sicherheit von IT-Systemen nach wie vor eines der grundlegenden Mittel zum Implementieren sicherer Systeme. Der Artikel beschreibt, wie beides – verschlüsselte Übertragung und verschlüsselte Speicherung – mit den Möglichkeiten der Oracle-Datenbank umgesetzt werden kann.

Die Ausfuhr von Verschlüsselungstechnologien in andere Staaten ist in vielen Ländern der Welt genehmigungspflichtig, so auch in der Bundesrepublik und in den USA. Bruce Schneier hat in diesem Zusammenhang in seinem Standardwerk zur Verschlüsselung „Applied Cryptography" bereits im Jahr 1996 darauf aufmerksam gemacht, dass in den USA diese Ausfuhr seiner Meinung nach nur genehmigt wird, wenn die Verschlüsselung von deren Geheimdiensten umgangen werden kann. Simon Singh bestätigt diese Aussage in seinem Buch „The Code Book" (1999) mit dem konkreten Hinweis auf Data Encryption Standard (DES). Es wird sich also auch bei dem Nachfolger von DES, dem Advanced Encryption Standard (AES), und anderen kommerziell verfügbaren Verschlüsselungsalgorithmen nicht anders verhalten.

Aber nur weil für Geheimdienste – das gilt sicherlich nicht allein für die USA – verschlüsselte Daten kein unüberwindbares Hindernis darstellen, bedeutet das nicht, dass Verschlüsselung überflüssig ist. Ganz im Gegenteil. Verschlüsselung bietet einen hohen Schutz vor:

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

In Kombination mit weiteren Sicherheitsmaßnahmen organisatorischer und tech-

nischer Art ist der Einsatz von Verschlüsselungstechnologien deshalb nach wie vor eine 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 Secure Sockets Layer (SSL) zu verschlüsseln. Zum anderen stehen prozedurale und deklarative Möglichkeiten zur Verfügung, um gespeicherte Daten zu verschlüsseln.

Daten im Netzwerk verschlüsseln

Mit der Freigabe von Oracle Database 12*c* R1 im Juni 2013 wurde die Verschlüsselung des Netzwerkverkehrs über SSL sowie nativ über SQL*Net, die bis dahin beide Bestandteil der Advanced Security Option (ASO) waren, als Feature der Datenbank verfügbar – und zwar sowohl für die Enterprise Edition als auch für die Standard Edition. Das gilt auch für vorangegangene Releases, zum Beispiel für Oracle Database 11*g*, sofern diese das technisch unterstützen. Dokumentiert ist die Änderung der Lizenzbedingungen in den Handbüchern zur Lizenzierung *(siehe „http://docs.oracle.com/cd/E16655_01/license.121/e17614/options.htm#DBLIC143“)*.

Entscheidet man sich für die native Möglichkeit, kann man die Verschlüsselung im einfachsten Fall über einen oder zwei Einträge in der Datei „SQLNET.ORA“ steuern. Die zu verwendenden Parameter heißen „SQLNET.ENCRYPTION_SERVER“ und „SQLNET.ENCRYPTION_CLIENT“. Der letztgenannte wird verwendet, wenn eine Datenbank in verteilten Umgebungen auch Client ist. Man setzt „SQLNET.ENCRYPTION_SERVER = REQUIRED“ und anschließend ist jeder Versuch, eine Verbindung zur Datenbank aufzubauen, nur noch dann erfolgreich, wenn diese Verbindung verschlüsselt ist. Neben „REQUIRED“ stehen für Server und Client weitere Einstellungen zur Verfügung:

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

Steht der Parameter beispielsweise auf „REQUESTED“, können die Einstellungen für die darauf zugreifenden Clients nach Wunsch unterschiedlich gesetzt sein. Während einzelne Clients über ihre „SQL*NET.ORA“-Dateien eine Verschlüsselung erzwingen („REQUIRED“), könnten andere, sofern das Sinn ergibt, darauf verzichten („REJECTED“). *Abbildung 1* zeigt die möglichen Parameter-Kombinationen und ihre Auswirkungen auf die Verschlüsselung.

Neben der Festlegung, ob verschlüsselt wird oder nicht, sind zusätzliche Einstellungen möglich. So kann der Algorithmus bestimmt werden, mit dem verschlüsselt wird (Parameter „sqlnet.encryption_types_server/_CLIENT“), sowie ob und welche Prüfsummenverfahren zu nutzen sind (Parameter „SQLNET.CRYPTO_CHECKSUM_xxx“). Ist der Verschlüsselungsalgorithmus nicht ausdrücklich festgelegt, vereinbaren Client und Server bei der ersten Kontaktaufnahme („handshake“) den Algorithmus zufällig aus der Reihe der gemeinsam zur Verfügung stehenden Algorithmen. Ist das Prüfsummenverfahren nicht aktiviert, findet keine Überprüfung statt. Wird es aktiviert, aber kein Verfahren benannt, vereinbaren Client und Server, wie bei der Verschlüsselung, während der Kontaktaufnahme ein Verfahren.

Die meisten Kunden, die die Netzwerk-Verschlüsselung einsetzen, verwenden wegen der einfachen Implementierung die gerade beschriebene native Variante. Aber es geht auch über SSL. Allerdings wird dem Datenbank-Administrator, der keine Erfahrungen mit dem Einsatz von SSL hat, hier das Prozedere deutlich aufwändiger erscheinen. Wenn man außerdem bedenkt, dass die Verschlüsselung über SSL eventuell mit zusätzlichen Kosten für Zertifikate verbunden ist, der Verbindungsaufbau geringfügig langsamer ist und dass je nach Architektur auch die Benutzer-Accounts noch anzupassen sind, wird klar, wer SSL überwiegend benutzt: Es sind Unternehmen, die in anderen Bereichen ebenfalls SSL standardmäßig einsetzen und die sich auf die noch etwas höhere Sicherheit verlassen, die SSL ihnen bietet. Das Implementieren von SSL erfolgt in folgenden Schritten:

- Zertifikate für Client und Server auf der Server- und der Client-Seite bereitstellen
- Auto-open-Wallet anlegen und Zertifikate importieren
- „SQLNET.ORA” anpassen (Parameter „WALLET_LOCATION” und „SSL_CLIENT_AUTHENTICATION”)
- „LISTENER.ORA“ (Server) und „TNSNAMES.ORA“ (Client) anpassen

Eine detaillierte Beschreibung der Vorgehensweise liefert der Oracle-Support in „Step by Step Guide To Configure SSL Au-

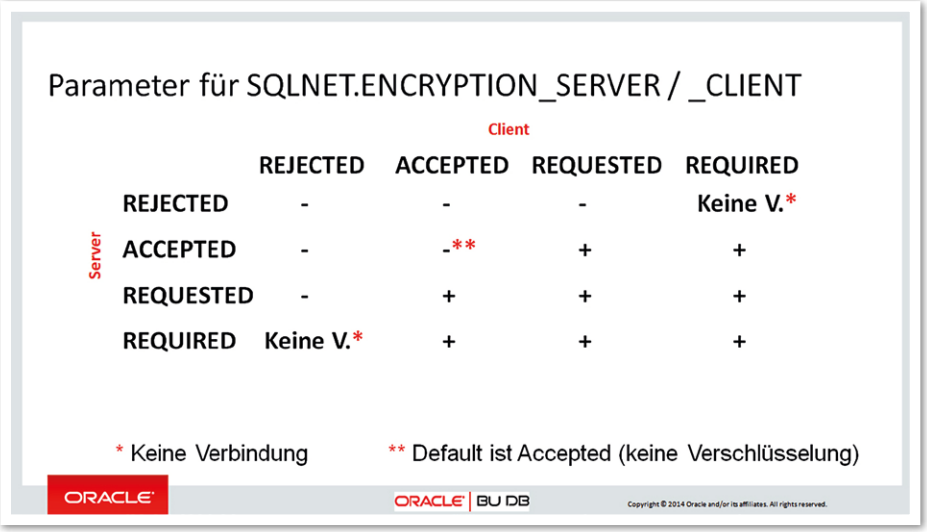

Abbildung 1: Einstellungen von SQLNET.ENCRYPTION_SERVER/_CLIENT, (– = unverschlüsselt, + = verschlüsselt)

```
CREATE OR REPLACE FUNCTION crypt (eingabe IN VARCHAR2)
RETURN RAW
IS
  v_rohdaten         RAW(200);      -- verschlüsselter Wert
  v_schluessel           RAW(32);       -- 256-bit Schlüssel
  v_verschluesselung PLS_INTEGER := -- Algorithmus
     DBMS_CRYPTO.ENCRYPT_AES256 +
     DBMS_CRYPTO.CHAIN_CBC +
     DBMS_CRYPTO.PAD_ZERO;
BEGIN
  SELECT schluessel INTO v_schluessel
  FROM schluesseltabelle
  WHERE sysdate BETWEEN startdatum AND nvl(enddatum, sysdate);
   -- Der zum Zeitpunkt gültige Schlüssel wird aus der
   -- (zuvor angelegten) Tabelle mit den Schlüsseln
   -- in die dafür vorgesehene Variable eingestellt.
  v_rohdaten   := DBMS_CRYPTO.ENCRYPT(
       src => UTL_I18N.STRING_TO_RAW (eingabe, 'AL32UTF8'),
       typ => v_verschluesselung,
       key => v_schluessel);
   -- Der Wert, der der Funktion mit dem Zeichensatz UTL_I18N -- übergeben, wird mit
dem angegebenen Algorithmus verschlüsselt, nach AL32UTF8 konvertiert und in die
   -- Variable v_rohdaten eingestellt.
RETURN v_rohdaten;
   -- v_rohdaten ist der Return Wert der Funktion.
END;
```

Listing 1

thentication“ (Doc-ID 736510.1). Für Anwendungen ist jede Form der Netzwerk-Verschlüsselung transparent, sie müssen also nicht angepasst werden.

Gespeicherte Daten prozedural verschlüsseln

Schon seit vielen Versionen der Datenbank gibt es die Möglichkeit, ohne zusätzliche Kosten Daten über Packages prozedural zu verschlüsseln. Die Packages sind in jeder Edition der Datenbank verfügbar und müssen von Anwendungen oder Triggern aufgerufen werden. Die Verwaltung der Schlüssel, die gemeinhin als kritischster Teil jeder Verschlüsselungsstrategie gilt, muss selbst organisiert werden.

Zwei Packages sind zu nennen: Zunächst „DBMS_OBFUSCATION_TOOLKIT“, das nur noch aus Gründen der Kompatibilität im Lieferumfang der Datenbanken enthalten ist und nicht mehr verwendet werden sollte. Das aktuell für die prozedurale Verschlüsselung verfügbare Package heißt „DBMS_CRYPTO“. Um einen Eindruck von der Arbeit mit diesem Package zu erhalten, zeigt *Listing 1* als Beispiel eine Funktion, die zum Verschlüsseln von Daten verwendet werden könnte. Zur Erläuterung sind Kommentare eingefügt.

Gespeicherte Daten deklarativ verschlüsseln

Seit der Datenbank-Version 10 bietet Oracle im Rahmen von ASO eine deklarative Methode an, um Daten zu verschlüsseln. Das Feature der Option ist „Transparent Data Encryption“ (TDE). Damit fallen zwar zusätzliche Lizenzgebühren an, aber der Erwerb einer Lizenz für ASO erlaubt unter anderem neben der Verschlüsselung von Benutzerdaten auch die Verschlüsselung von Backups mit RMAN und Exports mit Data Pump.

In Version 10 funktionierte TDE zunächst nur mit Tabellen-Spalten. Dabei gab und gibt es immer noch erhebliche Einschränkungen: So lassen sich bestimmte Datentypen und Fremdschlüsselspalten nicht verschlüsseln. Auch können keine Indizes auf verschlüsselte Spalten gelegt werden.

Schon in der Version 11 der Datenbank wurde TDE für das Verschlüsseln ganzer Tablespaces verfügbar. Diese Variante unterliegt keinerlei Einschränkungen. Weil das Ver- und Entschlüsseln hier im Rahmen der Ein-/Ausgabeoperation erfolgt, ist es auch noch performanter als die Spaltenverschlüsselung – und das nicht erst seit dem möglichen Rückgriff auf die Verschlüsselungstechnologien neuerer CPUs. Die Verschlüsselung von Tablespaces ist also in der Regel die empfohlene Variante. Ausnahmen von dieser Empfehlung betreffen allenfalls Datenbanken, in denen weniger als 5 Prozent der gespeicherten Daten verschlüsselt und auf die die ange-

```
ENCRYPTION_WALLET_LOCATION =
   (SOURCE = (METHOD = FILE)
      (METHOD_DATA = (DIRECTORY = /etc/wallets/orcl)))
```

Listing 2

```
ADMINISTER KEY MANAGEMENT
CREATE KEYSTORE  '/etc/wallet/orcl' IDENTIFIED BY einpasswort

Legt den „keystore“ ebenfalls mit dem Namen „ewallet.p12“ an

ADMINISTER KEY MANAGEMENT
SET KEYSTORE OPEN IDENTIFIED BY einpasswort

Öffnet den „keystore“

ADMINISTER KEY MANAGEMENT
SET KEY IDENTIFIED BY einpasswort WITH BACKUP

Erzeugt den Master Key
```

Listing 3

deuteten Einschränkungen nicht oder nie (wer kann das schon wissen?) zutreffen werden.

Das Verwalten und Erzeugen der Schlüssel im Rahmen von TDE erfolgt immer komplett durch die Datenbank. Dabei wird nicht zwischen der Spalten- und der 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 bei der Spalten-Verschlüsselung im Data Dictionary und bei der Tablespace-Verschlüsselung in den Headern der Datendateien gespeichert. Die zweite Stufe ist der sogenannte „Master Key“, der in der Literatur auch als „key-encryption key“ bezeichnet wird. Es handelt sich dabei um einen Schlüssel, der ausschließlich dazu dient, die Schlüssel der Tabellen und Tablespaces zu ver- und zu entschlüsseln. Auch dieser Schlüssel wird durch die Datenbank automatisch ohne irgendeine Möglichkeit der Einflussnahme durch den Anwender erzeugt.

Der Master Key ist immer außerhalb der Datenbank gespeichert. Dazu dient normalerweise eine kleine verschlüsselte Datei, die in Oracle Database 10*g* und 11*g* als „wallet“ und in Oracle Database 12*c* als „keystore“ bezeichnet wird. Alternativ kann der Master Key aber auch in einer speziellen Hardware, einem sogenannten „Hardware Security Modul“ (HSM), gespeichert sein. In der Datei „sqlnet.ora“ ist festgelegt, welche Variante für den Key genutzt wird.

Das Beispiel in *Listing 2* zeigt, wie der Master Key in einem „wallet/keystore“ angelegt und verwaltet wird und in welchem Verzeichnis dieses „wallet/keystore“ gespeichert ist. Obwohl es für unterschiedliche Betriebssysteme unterschiedliche Default-Einstellungen für die Speicherung des „wallet/keystore“ gibt, ist es empfehlenswert, den Ort der Speicherung explizit zu benennen.

Der Eintrag hat zunächst keinerlei Konsequenzen. Er ist auch identisch für Datenbanken der Versionen 10, 11 und 12. Zwar ist die weitere Vorgehensweise dann sehr ähnlich, allerdings wird in der Version 12 eine ganz unterschiedliche Syntax verwendet. In den Versionen 10 und 11 ist das Privileg „ALTER SYSTEM“ erforderlich. Dann führt der Befehl „ALTER SYSTEM SET ENCRYPTION KEY IDENTIFIED BY „einpasswort““ dazu, dass das Datenbank-System in dem in „sqlnet.ora“ angegebenen Verzeichnis eine verschlüsselte Datei mit dem Namen „ewallet.p12“ anlegt und dort den Master Key speichert. Dieser Master Key ist nicht identisch mit dem Passwort „einpasswort“.

Wird das Passwort ohne Anführungszeichen eingegeben, muss es später in Großbuchstaben eingegeben werden. Eine Änderung des Master Key ist über den gleichen „ALTER SYSTEM“-Befehl möglich. Bei der Änderung des Master Key werden lediglich die Tabellen- beziehungsweise Tablespace-Schlüssel neu verschlüsselt, was natürlich wenig zeitaufwändig ist. Der neue Master Key wird dann ebenfalls in der Datei „ewallet.p12“ gespeichert. Die alten Master Keys werden dabei nicht gelöscht, da man sie eventuell noch benötigt.

In der Version 12 ist entweder das Privileg „ADMINISTER KEY MANAGEMENT“ oder die Rolle „SYSKM“ erforderlich. Dann sind für diese Aktionen (Anlegen des „keystore“ und Erzeugen des Master Key) drei Befehle nötig *(siehe Listing 3)*.

Die Groß- und Kleinschreibung des Passwortes wird in Version 12 auch ohne Hochkommata berücksichtigt. Eine Änderung des Master Key ist über den Befehl „ADMINISTER KEY MANAGEMENT“ ebenfalls möglich. Zugriff auf verschlüsselte Daten erhält man nur über geöffnete „wallet/keystore“. Das Öffnen erfolgt bei jedem Start einer Datenbank im Mount-Status, bei der Version 10 oder 11 über den Befehl „ALTER SYSTEM SET ENCRYPTION WALLET OPEN IDENTIFIED BY „einpasswort““. Das Pendant dieses Befehls für die Datenbank Version 12 lautet „ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN IDENTIFIED BY „einpasswort““

```
ADMINISTER KEY MANAGEMENT
CREATE (LOCAL) AUTO_LOGIN KEYSTORE
FROM KEYSTORE '/etc/wallet/orcl' IDENTIFIED BY einpasswort
```

Listing 4

```
ADMINISTER KEY MANAGEMENT
ALTER KEYSTORE PASSWORD
IDENTIFIED BY einpasswort SET einneuespasswort WITH BACKUP
```

Listing 5

Nach dem Öffnen des „wallet/keystore“ wird der Master Key ausgelesen und in der SGA in leicht verschlüsselter Form („obfuscated“) vorgehalten. Das „wallet/keystore“ selbst wird deshalb von der laufenden Datenbank nicht mehr benötigt – was zum Beispiel die Möglichkeit eröffnet, sie auf einem Stick zu speichern und diesen nach dem Öffnen der Datenbank vom Rechner abzuziehen sowie an einer sicheren Stelle aufzubewahren.

Es ist möglich, „wallet/keystore“ bei jedem Start der Datenbank automatisch zu öffnen. Man spricht dann von „auto open“ oder „auto login wallets/keystores“. Damit deren Diebstahl bei gleichzeitigem Entwenden der Datendateien nicht zu einem Sicherheitsrisiko wird, können sie auch als sogenannte „local auto open“ oder „local auto login wallets/keystores“ angelegt werden. Diese lassen sich dann nur auf dem Rechner öffnen, auf dem sie angelegt wurden.

Das automatische Öffnen des „wallet“ erfolgt in der Version 10 oder 11 mit dem Oracle Wallet Manager (owm) oder über das Werkzeug „ORAPKI“. Es wird dann im selben Verzeichnis, in dem auch die Datei „ewallet.p12“ liegt, eine Datei „ewallet.sso“ angelegt, in der der Master Key der Datenbank verfügbar gemacht wird. Mit dem „owm“ beziehungsweise über „orapki“ ist auch das Passwort für das Wallet zu ändern.

In der Version 12 der Datenbank sind keine besonderen Werkzeuge für diese beiden Aktionen nötig. Hier gibt es dazu SQL-Befehle. Das automatische Öffnen des „keystore“ wird veranlasst über den Befehl in *Listing 4*, während das Passwort mit dem Befehl in *Listing 5* geändert wird.

„wallet/keystore“ sollten immer gesichert werden, nachdem sich Passwort oder Master Key geändert haben. In der Version 12 erfolgt das automatisch durch die Verwendung der Klausel „WITH BACKUP“ bei den entsprechenden Befehlen. In der Version 10 oder 11 muss der für das „wallet“ verantwortliche Mitarbeiter selbst für dieses Backup sorgen. Das ist extrem wichtig, denn es gibt keine Möglichkeit, verschlüsselte Daten ohne den Master Key aus dem „wallet/keystore“ zu entschlüsseln.

Auch im Rahmen des normalen Backups der Datenbank sollte man sich eine Strategie für das Sichern des „wallet/keystore“ überlegen. Vor allem bei Verwendung von „auto login-wallets/keystores“ wird dringend davon abgeraten, diese zusammen mit dem Backup abzulegen. Der Diebstahl des Backups würde dazu führen, dass der Dieb Zugriff auf die verschlüsselten Daten erlangen kann.

```
CREATE TABLESPACE sicheristsicher
DATAFILE '/app/oracle/oradata/dateiname.dbf' SIZE 10G
ENCRYPTION USING 'AES128' DEFAULT STORAGE (ENCRYPT)
```

Listing 6

Abschließend ist nur noch zu klären, wie man verschlüsselte Daten in einem Tablespace erzeugt. Das Verfahren ist identisch für die unterschiedlichen Datenbank-Versionen. Zunächst ist zu beachten, dass ein Tablespace nicht nachträglich verschlüsselt werden kann, sondern immer als verschlüsselt angelegt sein muss. Vorhandene Daten müssten also mit einem „CREATE TABLE AS SELECT“, mit einem Export/Import oder anderen Verfahren in ein verschlüsseltes Tablespace verschoben werden. *Listing 6* zeigt, wie zum Beispiel ein verschlüsseltes Tablespace angelegt wird.

Alle Daten, die in das Tablespace eingefügt beziehungsweise dort manipuliert werden, sind unter Beibehaltung der bekannten Befehle ohne weitere Klauseln (in der Oracle-Terminologie „transparent“) ver- beziehungsweise entschlüsselt. Eine Änderung von Anwendungen ist hier also ebenso unnötig wie beim Verschlüsseln des Netzwerk-Verkehrs. Das gilt auch für alle gängigen größeren Anwendungen von Oracle, zum Beispiel die E-Business Suite. Für SAP-Anwendungen gibt es Support-Hinweise, wie die Systeme aufzusetzen sind, um TDE zu nutzen.

Je nach Datenbankversion sind unterschiedliche Verschlüsselungsalgorithmen verfügbar. Nur diese können verwendet werden; eigene oder Open-Source-Verfahren wie Blowfish sind nicht möglich.

Während Tabellenspalten mit dem Befehl „ALTER TABLE“ nachträglich verschlüsselt oder auch „umgeschlüsselt“ werden können beziehungsweise die Verschlüsselung auch wieder komplett aufgehoben werden kann, ist eine nachträgliche Änderung oder Aufhebung der Verschlüsselung eines Tablespace nicht möglich. Auch eine nachträgliche Schlüssel-Änderung ist nicht erlaubt.

TDE und Container-Datenbanken

Zum Arbeiten mit Container-Datenbanken in der Version 12 sind ein paar zusätzliche Hinweise wichtig. Es gibt für eine Container-Datenbank immer nur einen „keystore“. Darin speichern alle „pluggable databases“ ihre Master Keys. Diese 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. Schließlich ist bemerkenswert, dass ein „keystore“ immer erst auf der Ebene der Container-Datenbank geöffnet werden muss, bevor einzelne „pluggable databases“ ihn für sich öffnen und nutzen können.

Schlussbemerkung: Bei diesem Beitrag handelt es sich um die ausformulierte Version eines Vortrags, der am 3. Juni 2014 anlässlich der DOAG 2014 Datenbank in Düsseldorf gehalten wurde.

Heinz-Wilhelm Fabry
heinz-wilhelm.fabry@oracle.com