Protecting Sensitive Information with Database Encryption

Rochester, NY OWASP

Ralph Durkee

Durkee Consulting, Inc.

rd@rd1.net

What is covered

- Provides specific technical recommendations for:
  - When to encrypt information
  - How to avoid encryption
  - Where in the architecture to encrypt
  - Key Management
  - Generic architecture and encryption techniques
  - Implementation for specific Databases
    - Oracle 8g-9g
    - Oracle 10-11i
    - MS SQL Server 2005

Ralph Durkee

- **Rochester OWASP President** since 2004
- **Founder Durkee Consulting** since 1996
- **Application Security**, development, auditing, PCI compliance, pen testing and consulting
- **CIS (Center for Internet Security)** – Helped development of benchmark standards – Linux, BIND DNS, OpenLDAP, FreeRadius, Unix, FreeBSD
- **SANS Community Instructor** and course developer
- Rochester Area Security Consulting, Ethical Hacking and Auditing

When?
Requirements for Encryption

May be used to help meet regulations:

- **GLBA** to ensure confidentiality of customer records and information
- **State Breach Notification Laws -** require notification if information was not encrypted
- **Regulatory efforts** impose stiffer fees and fines in the event that a breach occurs and steps are not taken to appropriately protect sensitive data

When?
Requirements for Encryption

Payment Card Industry, PCI DSS Requirement 3.4

Requirement 3: Protect stored cardholder data

3.4 Render PAN, at minimum, unreadable anywhere it is stored (including data on portable digital media, in logs, and data received from or stored by wireless networks) by using any of the following approaches:

- Strong one-way hash functions (hashed indexes)
- Truncation
- Index tokens and pads (pads must be securely stored)
- Strong cryptography with associated key management processes and procedures.

When?
Requirements for Encryption

Payment Card Industry, PCI DSS Requirement 3.4 *(continued)*

3.4.1 If disk encryption is used (rather than file- or column-level database encryption), logical access must be managed independently of native operating system access control mechanisms (for example, by not using local system or Active Directory accounts). Decryption keys must not be tied to user accounts.

Payment Card Industry, PCI DSS Requirement 3.5

3.5 Protect encryption keys used for encryption of cardholder data against both disclosure and misuse.

3.5.1 Restrict access to keys to the fewest number of custodians necessary

3.5.2 Store keys securely in the fewest possible locations and forms.

When?
Requirements for Encryption

PCI DSS Requirement 3.6

3.6 Fully document and implement all key management processes and procedures for keys used for encryption of cardholder data, including the following:

3.6.1 Generation of strong keys

3.6.2 Secure key distribution

3.6.3 Secure key storage

3.6.4 Periodic changing of keys

- As deemed necessary and recommended by the associated application (for example, re-keying); preferably automatically
- At least annually.

When?
Requirements for Encryption

PCI DSS Requirement 3.6 *(continued)*

3.6.5 Destruction of old keys

3.6.6 Split knowledge and establishment of dual control of keys (so that it requires two or three people, each knowing only their part of the key, to reconstruct the whole key)

3.6.7 Prevention of unauthorized substitution of keys

3.6.8 Replacement of known or suspected compromised keys

3.6.9 Revocation of old or invalid keys

3.6.10 Requirement for key custodians to sign a form stating that they understand and accept their key-custodian responsibilities.

IMPORTANT: PCI States CANNOT store Mag. Strip, PIN or Card Verification Code

Why? - Threats, Risks and Rational for Data Encryption

- **Spectrum of Threats to DB Information**
  - T1 - Theft or Loss of DB System or Storage Media
  - T2 - Theft or Loss of DB Backup Media
  - T3 - Theft or Loss of OS Backup Media
  - T4 - Exploitation of Direct DB Access
  - T5 - DB System Compromise
  - T6 - DBA Insider or DBA Account Compromise
  - T7 - Application or DB User Account Compromise
  - T8 - Application or Middleware OS Compromise
  - T9 - Exploitation of Application with Restricted Account
  - T10 - Exploitation of Application with Unrestricted Account

Encryption Terms

- Encryption Algorithm
- Key
- Clear Text
- Cipher Text
- Initialization Vector
- Secure Hash (0 keys)
- Symmetric Encryption (1 key)
- Asymmetric or Public Key Encryption (2 keys)

Real World Public Key Implementations

Problems:

- Asymmetric encryption 100 times slower
- Symmetric encryption
  - Requires a shared secret
  - Doesn’t scale well
  - Requires (N*(N-1))/2 keys for N people

Real World Implementations

- Use a hybrid of symmetric, asymmetric & hash
- Provides both good performance and scalability
- Doesn’t required a pre-shared secret.
- Examples: PGP, SSL/TLS, IPSec, S/MIME

Encryption Alternatives

Before encrypting information ask:

- Does the sensitive information really need to be stored?
- Can the sensitive information be truncated?
- Can the sensitive information be stored as a salted, secure hash?
- Can the sensitive information be stored only in memory, or stored temporarily and then securely wiped?

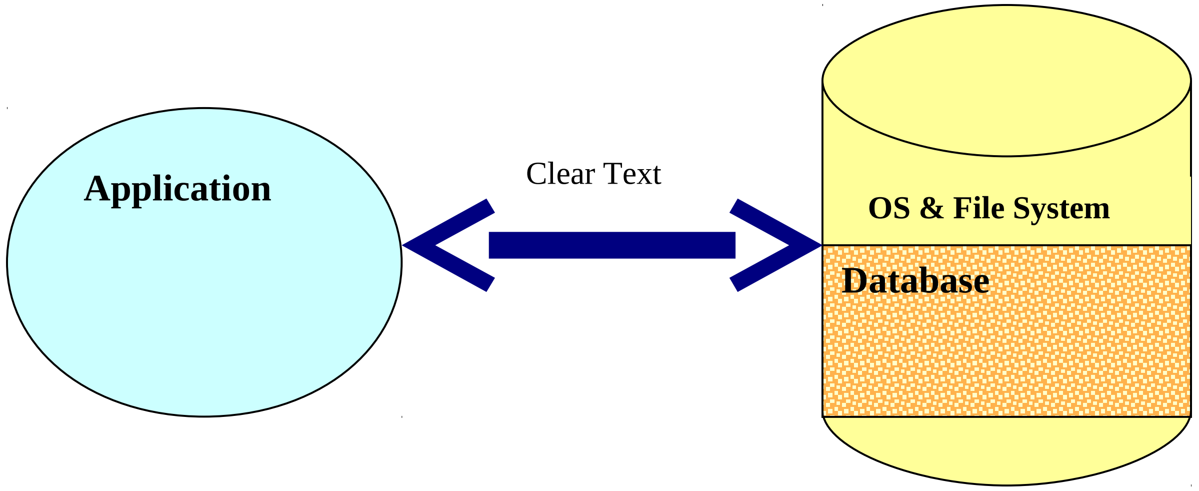
Home Grown Encryption

- Bad idea to invent encryption algorithms
- Do not accept proprietary encryption
- Acceptable algorithms are very difficult and require:
  - Invented by professional cryptologist
  - Subject to years of open analysis and scrutiny
- Many past failures by the brightest and well funded
  - MD4 hash by Ronald Rivest
  - Helix by Bruce Schneier
  - LANMAN Hash by Microsoft
  - DVD CSS (Content Scrambling System)

Standard Symmetric Algorithms

Algorithm	Type	Key	Strength	DBMS
DES	Block	56	Weak	Oracle, MS SQL
3DES	Block	128	Acceptable	Oracle, MS SQL
AES-192	Block	192	Strong	Oracle DBMS Crypto, MS SQL
AES-256	Block	256	Strong	Oracle DBMS Crypto, MS SQL
RC4	Stream	1-256	Strong	Oracle DBMS Crypto, MS SQL
RC2	Block	128	Acceptable	MS SQL

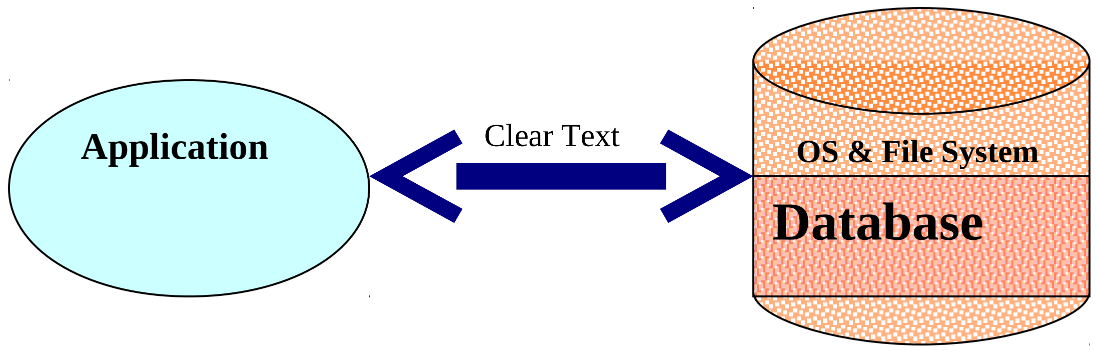
Full DB Encryption



- Lack of Granular Access Control
- Performance Impact
- Limited Key Management

✷ Not Recommended

OS or File System Encryption

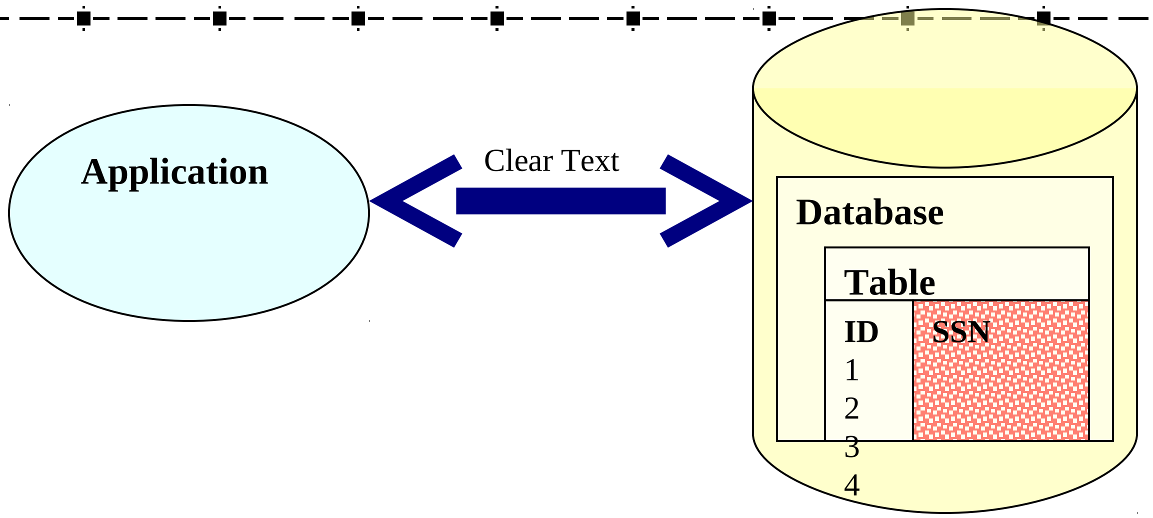


Same problems as Full DB Encryption

- Lack of Granular Access Control
- Performance Impact
- Limited Key Management

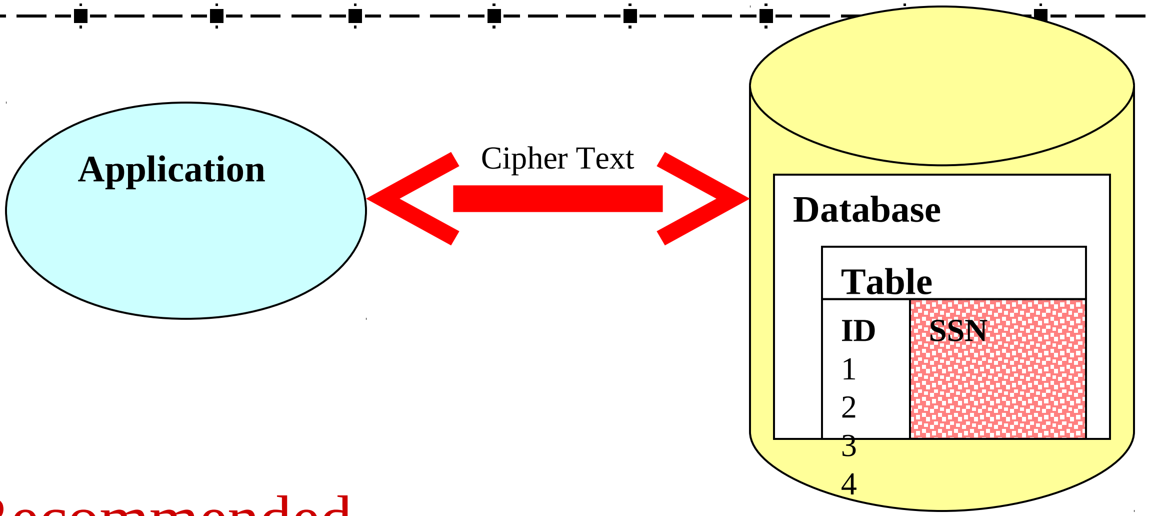
✷ Not Recommended

Field Level by DB or Middleware



- Recommended
- + Granular Access Control
- + Limited Performance Impact
- - Clear text communications should be encrypted

Field Level by the Application



- Recommended
- + Granular Access Control
- + Resistant to DB attacks and DBA Insider threats
- + Less impact from other weak applications
- ± Each application implements key management

Application Level Passwords and Authentication

- User Passwords stored as salted secure hash
  - Salt of 8 or more random characters
  - Unique salt for each password
  - Stored in clear with password
  - Example: $1$i0/B42.e$GcsbF4Y0xb0PM2Um8jIoI1$
- Application Passwords
  - Used to authenticate applications to other systems.
  - Must be retrievable by the application
  - Stored with symmetric or public key encryption.
  - Same issues as key management

Key Management Architectures

- Not Recommended
  - K0 - Storing and Hiding Keys in Software
  - K1 - Auto Decryption by OS or FS
  - K2 - Auto Decryption by DBMS
  - K3 - Key Stored in DB and Readable by DBA
- Recommended
  - K4 - Key Stored in FS on DB Server and Readable by DBA
  - K5 - Key Stored in FS on DB Server and NOT Readable by DBA
- Preferred
  - K6 - Key Stored on the Client, Application Or Mid-Tier Server with Minimal Access

Key Management Threat Matrix

	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10
K1	Y1	N	N	N	N	N	N	N	N	N
K2	N	N	N	N	N	N	N	N	N	N
K3	N	N	N	Y3	N	N	Y	Y	Y	N
K4	N	Y	N	Y3	N	N	Y	Y	Y	P4
K5	N	Y	N	Y	N	Y	Y	Y	Y	P4
K6	Y	Y	Y	Y	Y2	Y2	Y	N	Y	P4

Key Management Threat Matrix Footnotes –Explanations

- **Yes 1 (T1 vs. K1) -** Yes, if …
  - Assumed that the key is not stored in clear text on the system
  - With weak protection such as MS Windows LANMAN hash.
  - Protection is only as good as protection for the key
  - See section 3.1 for details
- **Yes 2 (T6 & T7 vs. K6) -** Yes if …
  - Key is not sent to DBMS for decryption
- **Yes 3 (T4 vs. K3 & K4) -** Yes, If …
  - Key is readable only by a single application account and the DBA.
  - Account access does not include Admin level access.
- **Partial 4 -** Yes, in many cases, but not if …
  - decryption is automated by the application

Encryption for Oracle 8i-11g

- Oracle DBMS Obfuscation Toolkit (DOTK)
  (Only option for older Oracle 8g & 9g)
- Oracle DBMS_CRYPTO package
  (Recommended Solution)
- Oracle Transparent Data Encryption (TDE)
- Oracle Advanced Security Option
- Oracle Database Vault

Oracle DOTK Checklist

1. Use only the 3DES encryption rather than DES
2. Avoid for highly sensitive information, Use DBMS_CYRPTO when available
3. Use a randomly generated key of at least 128 bits generated from DES3GetKey().
4. Use a good source of entropy for the random seed used for generating the key such as /dev/random on Unix/Linux systems and CryptGenRandom() on MS windows.
5. Use a randomly generated IV (Initialization vector) of 8-16 bytes (64-128 bits) for each encrypted record.

Oracle DBMS Obfuscation Toolkit

DOTK Encryption Procedure:

```
DBMS_OBFUSCATION_TOOLKIT.DES3Encrypt( inpu
   t_string IN VARCHAR2,
   key_string IN VARCHAR2,
   encrypted_string OUT VARCHAR2,
   which IN PLS_INTEGER DEFAULT TwoKeyMode,
   iv_string IN VARCHAR2 DEFAULT NULL);
```

- Also function with output returned
- Also function & procedures with **raw** parameters
- Default Null IV is Dangerous, should be random!

Oracle DBMS Obfuscation Toolkit

DOTK Decryption Procedure:

```
DBMS_OBFUSCATION_TOOLKIT.DES3Decrypt(
  input_string   IN   VARCHAR2,
  key_string     IN   VARCHAR2,
  decrypted_string  OUT  VARCHAR2,
  which          IN PLS_INTEGER DEFAULT TwoKeyMode
  iv_string      IN   VARCHAR2 DEFAUTL NULL);
```

- Also function with output returned
- Also function & procedures with **raw** parameters
- Need the same IV to decrypt.

Oracle DBMS Obfuscation Toolkit

DOTK DES3 Generate Key Procedure:

```
DBMS_OBFUSCATION_TOOLKIT.DES3GetKey(
 which    IN PLS_INTEGER DEFAULT TwoKeyMode,
 seed_string  IN   VARCHAR2,
 key          OUT  VARCHAR2);
```

- Also function with output returned
- Also function & procedure with **raw** parameters
- Important to use Random seed.

Oracle DBMS Crypto Checklist

1. Use either the AES192 or AES256 algorithm
2. Use DBMS_CRYPTO.RANDOMBYTES() to generate random keys, not DBMS_RANDOM
3. Use CBC (Cipher Block Chaining) mode.
   CFB Cipher Feedback Mode and
   OFB Output Feedback Mode are both ok
4. Do not use ECB Electronic Codebook chaining mode (It is weak)
5. Use PKCS5 for cryptographic padding rather than null padding

Oracle DBMS Crypto Encryption

Sample Encrypt function

```
DBMS_CRYPTO.ENCRYPT(
 src IN RAW,
 typ IN PLS_INTEGER,
 key IN RAW,
 iv  IN RAW DEFAULT NULL)
RETURN RAW;
```

- Also procedure with output as a parameter
- Important to use Random IV.
- Decrypt function & procedure are very similar.

Oracle DBMS Crypto Encrypt TYP Parameter

- **TYP** parameter specifies algorithms and modifiers

Feature	Options
Crypto algorithm	DES, 3DES, AES128, AES192, AES256, RC4, 3DES_2KEY
Padding forms	PKCS5, NONE, ZERO
Block Cipher chain mode	CBC, CFB, ECB, OFB

Oracle DBMS Crypto Encryption

DBMS Crypto Generate Random Bytes:

```
DBMS_CRYPTO.RANDOMBYTES (
   number_bytes IN POSITIVE)
 RETURN RAW;
```

- Use for Random IV and to generate random keys
- Do not use DBMS_RANDOM, as it’s weak.

Encryption for MS SQL Server 2005

Key Management Hierarchy

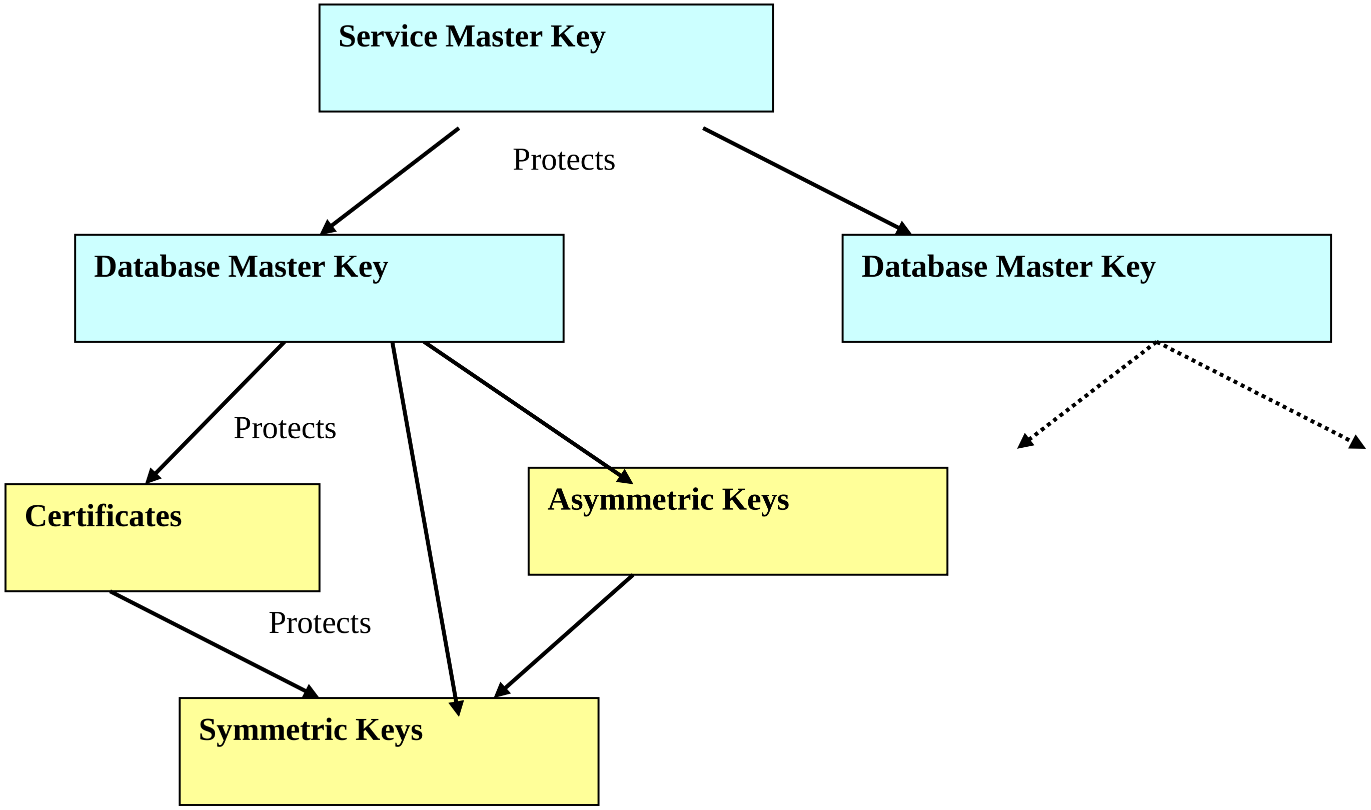
- **Service Master Key**
  - Root – Top Level Key – symmetric key
  - One Service Master key per installation
  - Auto-Generated at time of installation
  - Can not be directly access,
  - Can be regenerated and exported
  - Accessed by SQL Server Service account

MS SQL Server Key Management Hierarchy

Key Management Hierarchy

- **Database Master Keys**
  - Symmetric key
  - One Database Master key per Database
  - Used to encrypt all user keys in the database
  - Copy stored encrypted with Service Master Key
  - Also stored encrypted with a password
  - Recommend removing copy stored with service master key if possible.
- **User Keys**
  - May be a certificates, asymmetric keys or symmetric key
  - Generated as needed by DBA or users
  - Stored encrypted with Database Master key

MS SQL Server Key Management Hierarchy



MS SQL Server Protecting Password

- Avoid storing passwords if possible
- Use LDAP or Active Directory is possible
- Otherwise use Crypto API to generate secure salted hash:
  - *CryptGenRandom()* generates a salt
  - *CryptCreateHash()* creates hash object
  - *CryptHashData()* generated hash

MS SQL Server Checklist

1. Use either the AES192 or AES256 algorithm
2. Use randomly generated keys and passwords
   - generated by MS SQL Server,
   - or via CryptGenRandom() from MS Crypto API

1. Avoid storing keys or passwords in software
2. Remove the service key encrypted copy of the database master, if possible to reduce risk.

Summary

- Database Encryption is increasingly an important and more often required security control
- Effective layer of defense if properly implemented
- Protecting Keys hard to do, and is most common weakness
- Key Management and Encryption requirements needs to be defined early in application requirements
- Use these guidelines during the architecture, design and implementation of encryption.

Resources & References

Recommended MS SQL Server References

- **Protect Sensitive Data Using Encryption in SQL Server 2005**

http://www.microsoft.com/technet/prodtechnol/sql/2005/sqlencryption.mspx

Resources & References (2)

Recommended Oracle References

- **Encrypt Your Data Assets** - Scenario and example of DBMS_CRYPTO usage. http://www.oracle.com/technology/oramag/oracle/05-jan/o15security
- [Kornbrust] **2005 BlackHat Presentation on how to circumvent Oracle DB Encryption**. - http://www.blackhat.com/presentations/bh-usa-05/bh-us-05-kornbrus
- **Oracle Security Guide 16 “Developing Applications Using Data Encryption”** http://download.oracle.com/docs/cd/B14117_01/network.101/b10773/apdvncrp.htm#1006258

Thank You!

- Questions?