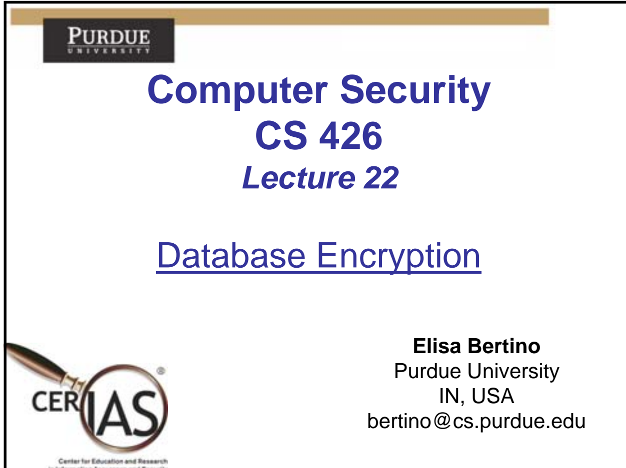


The problem

- How to protect the confidentiality of sensitive data in the database?
- Sensitive data examples: credit card numbers, medical data, new product specifications, etc.
- Possible risks to data confidentiality:
  - Use of weak authorization (for example weak or blank passwords) leading to access to confidential information (for example, payrolls) by unauthorized persons
  - Misconfiguration of access control
  - Authorized backdoors into the database (read-only accounts, non-production databases, backups)
  - Database administrators can access, inadvertently or maliciously, online data and backup data
  - SQL injection attacks through a poorly coded Web application

The problem attackers & protection goals

- Who could be the attackers? How to make their attacks more difficult (aka protection goal)?
  - System and database administrators:
    - They may have full access to everything in their administrative domain
    - Protection goal: make it difficult and time-consuming for them to read confidential information; use separation of duties; employment screening
  - Development staff:
    - They have an intimate knowledge of the code; they often obtain troubleshooting read-only rights to the production database to deal with emergency production problems
    - Protection goal: prevent compromise of the data even when they have access to the database

The problem attackers & protection goals

- Who could be the attackers? How to make their attacks more difficult (aka protection goal)?
  - Network intruders:
    - A network intruder is an unauthorized person who has gained access to the network. (S)he may attempt to eavesdrop the communication (for example those btw the application and the Database server) to gather confidential information or authentication credentials, or might attempt to crack the application.
    - Protection goal: encrypt communication and use application-level controls

PURDUE UNIVERSITY

The problem attackers & protection goals

- Who could be the attackers? How to make their attacks more difficult (aka protection goal)?
  - Application crackers:
    - They try to circumvent application security to gain unauthorized access. They can be considered like unauthorized users, but they may also be able to impersonate a legitimate user.
    - Worst case: the cracker gains administrative privileges
    - Protection goal: make access to the database “difficult” and time-consuming; mitigate SQL injections
  - Legitimate users:
    - (s)he may try to elevate his privileges, or to impersonate another legitimate user
    - Protection goal: strong authentication controls

PURDUE UNIVERSITY

The problem attackers & protection goals

- Who could be the attackers? How to make their attacks more difficult (aka protection goal)?
  - “traditional thieves”:
    - They might steal the database or the backup media
    - While database servers are typically kept in locked and limited-access data centres (physical security), backup media might leave the premises and are more exposed to theft

PURDUE UNIVERSITY

External requirements

- Legislations requiring the protection of data confidentiality:
  - Health Insurance Portability & Accountability Act (HIPPA)
  - Sarbanes-Oxley Act (SOX)
  - Gramm-Leach-Biley Act (GLBA)
  - Children’s Online Privacy Protection Act (COPPA)
- Business Compliance

PURDUE UNIVERSITY

External requirements

Health Insurance Portability & Accountability Act (HIPPA):

- It requires data safeguards that protect against “intentional or unintentional use or disclosure of protected health information “, and
- it mandates “to ensure the confidentiality, integrity and availability of all electronic protected health information the covered entity creates, receives, maintains, or transmits”
- It mandates “to implement a mechanism to encrypt and decrypt electronic protected health information”

External requirements

Business Compliance:

- Payment Card Industry (PCI) Data Security Standard
  - Stored cardholder data must be rendered unreadable, and it includes cryptographic methods in the recommended controls
  - Adopted by American Express, Visa, MasterCard and several other payment card companies

The solution

- We have already discussed authentication and access control as means to allow access to the data to authorized persons only
- However, authentication & access control may not be enough (DB administrators can still access and see the data)
- If data are sensitive it is also possible to encrypt them
  - Data encryption is the last barrier to protect sensitive data confidentiality

Encrypting the database

- ❖ Which type of encryption (symmetric or asymmetric)?
- ❖ Encryption vs. Obfuscation
- ❖ Cryptographic risks
- ❖ What should be encrypted?
- ❖ Which component should perform the encryption?

Which type of encryption?

- Symmetric key cryptography
  - DES, AES
  - Faster than asymmetric cryptography
- PROs
  - Performance
- CON's
  - Key management:
  - Since the same key is used both to encrypt and decrypt, the key must be distributed to every entity that needs to work with the data
  - If the key is obtained by an attacker, then confidentiality (and integrity) of data are at risk
  - Once the key is at the decrypting location, it must be secured so that an attacker can not steal it

PURDUE UNIVERSITY

Which type of encryption?

- Asymmetric key (i.e., public-private key) cryptography
- The keys used to encrypt and decrypt the data are different. This doesn't require a shared secret, BUT
- It still requires the owner of the keys to keep secret the private key

PURDUE UNIVERSITY

Obfuscation

- In cryptography, obfuscation refers to encoding the input data *before* it is sent to a *hash function* or other encryption scheme.
- This technique helps to make brute force attacks unfeasible, as it is difficult to determine the correct cleartext

PURDUE UNIVERSITY

Obfuscation

- In certain cases, obfuscation would be preferable to encryption
- Example: an audit report on a medical system
  - This report may be generated for an external auditor, and contain sensitive information. The auditor will be examining the report for information that indicates possible cases of fraud or abuse.
  - Assume that the management has required that Names, Social Security Numbers and other personal information should not be available to the auditor except on an as needed basis.
  - The data needs to be presented to the auditor, but in a way that allows the examination of all data, so that patterns in the data may be detected.
  - Encryption would be a poor choice in this case, as the data would be rendered into ASCII values outside of the range of normal ASCII characters. This would be impossible to read.
  - ➢ A better choice might be to obfuscate the data with a simple substitution cipher. While this is not considered encryption, it may be suitable for this situation.

PURDUE UNIVERSITY

Transposition Ciphers example Rail Fence cipher

- **STEP1: Write message letters out diagonally over a number *n* of rows**
- **STEP 2: Then read off cipher row by row**
- **Example**
  - **Original Message**: `meet me after the toga party`
- STEP1 trasforms it into:

```
m e m a t r h t g p r y
 e t e f e t e o a a t
```

- STEP2: ciphertext

```
MEMATRHTGPRYETEFETEOAAT
```

Obfuscation

Example of an obfuscation function using transposition (in Oracle PL/SQL):

```
create or replace package body obfs
as
xlate_from varchar2(62) :=
'0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz';
 xlate_to varchar2(62) :=
'nopqrstuvwxyz0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklm';
function obfs ( clear_text_in varchar2 ) return varchar2
        is begin
                return translate( clear_text_in, xlate_from, xlate_to )
        end;
function unobfs ( obfs_text_in varchar2 ) return varchar2
is begin
   return translate( obfs_text_in, xlate_to, xlate_from );
  end;
end;
```

NOTE1 This obfuscation is reversible, that is, it is possible to generate the cleartext data from the obfuscated one

NOTE2 The Oracle **Translate** function replaces a sequence of characters in a string with another set of characters. It replaces the 1st character in the *string_to_replace* with the 1st character in the *replacement_string*. Then it replaces the 2nd character in the *string_to_replace* with the 2nd character in the *replacement_string*, and so on.

Obfuscation

- Another obfuscation technique: masking
- It is useful in situations where it is only necessary to *display a portion* of the data
- Examples: the receipts printed at gas stations and convenience stores, the e-receipt from Expedia.
  - the last 4 digits of the credit are often displayed as clear text, while the rest of the credit card number has been masked with a series of X's.
- This is different from the previous technique in that the clear text cannot be reconstructed from the displayed data

Cryptographic risks

- Main risk: lost keys
- When the encryption key is lost, there is no "undelete" or "data recovery" program that can undo the encryption!

Key management is fundamental

- If an attacker can access the key, or insert a known key into the system, the encryption is broken
- If the key generation routines do not use "enough random" numbers, the attacker can guess the key
- All in all, the cryptographic infrastructure must be designed and implemented correctly

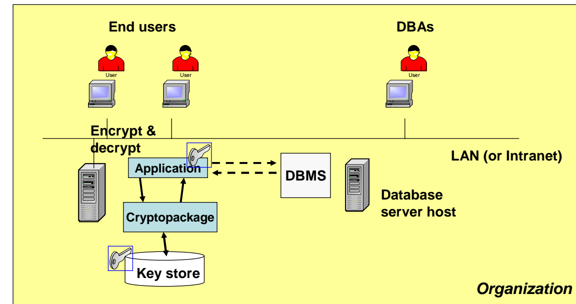
What to encrypt? *granularity*

- The full database (i.e. all tables)
- Cells (i.e., the value of a specific row or field within a row)

Where should encryption/decryption be performed?

- The application must request encryption and decryption.
  - CASE1 - In this case, keys are managed outside the DBMS (i.e. encrypt/decription performed by some package)
  - CASE2 - In this case, keys are managed by the DBMS (i.e. encrypt/decription performed by functions provided by the DBMS); however encryption/decryption has to be required by the application
  - CASE3 - Encryption and decryption and key management are performed by the DBMS engine "automatically"

CASE 1



CASE 1 Encryption at the application level

Application-level encryption (naïve solution):

- the database application developer uses an existing encryption library and embeds the key in the code
- Keys are generated outside the DBMS (i.e. by the encryption library).
- Hence the DBMS does not know the encryption keys

CASE 1 Encryption at the application level

- The application encrypts data before inserting them in the DB. Schematically:
  - Key = Cryptopackage.generatekey(param)
  - Encdata = Cryptopackage.Encrypt(data, key, algo)
  - SQL INSERT Encdata
- The application decrypts data after having read them from the DB
  - SQL SELECT data from Table
  - Cryptopackage.Decrypt data

PURDUE UNIVERSITY

CASE1 example (Javax.crypto)

1 – generate key

```
import javax.crypto.KeyGenerator;
import java.security.Key;
import java.security.NoSuchAlgorithmException;
import java.security.Security;
public class DESKeyGenerator {
    public static void main(String[] args) {
        Security.addProvider(new com.sun.crypto.provider.SunJCE());
        try {
            KeyGenerator kg = KeyGenerator.getInstance("DES");
            Key key = kg.generateKey();
            System.out.println("Key format: " + key.getFormat());
            System.out.println("Key algorithm: " + key.getAlgorithm());
        }
        catch (NoSuchAlgorithmException e) {
            e.printStackTrace();
        }
    }
}
```

PURDUE UNIVERSITY

CASE1 example (Javax.crypto)

2 – generate a cipher (provider)

```
import javax.crypto.Cipher;
import javax.crypto.NoSuchPaddingException;
import java.security.Security;
import java.security.NoSuchAlgorithmException;
public class DESCipherGenerator {
    public static void main(String[] args) {
        Security.addProvider(new com.sun.crypto.provider.SunJCE());
        try {
            Cipher cipher = Cipher.getInstance("DES");
            System.out.println("Cipher provider: " + cipher.getProvider());
            System.out.println("Cipher algorithm: " + cipher.getAlgorithm());
        }
        catch (NoSuchAlgorithmException e) {
            e.printStackTrace();
        }
        catch (NoSuchPaddingException e) {
            e.printStackTrace();
        }
    }
```

PURDUE UNIVERSITY

CASE1 example (Javax.crypto)

3 – encrypt data

```
public class DESCryptoTest {
    public static void main(String[] args) {
        Security.addProvider(new com.sun.crypto.provider.SunJCE());
        try {
            KeyGenerator kg = KeyGenerator.getInstance("DES");
            Key key = kg.generateKey();
            Cipher cipher = Cipher.getInstance("DES");

            byte[] data = "Hello World!".getBytes();
            System.out.println("Original data : " + new String(data));

            cipher.init(Cipher.ENCRYPT_MODE, key);
            byte[] result = cipher.doFinal(data);
            System.out.println("Encrypted data: " + new String(result));
```

PURDUE UNIVERSITY

CASE1 example (Javax.crypto)

3 – decrypt data

```
cipher.init(Cipher.DECRYPT_MODE, key);
byte[] original = cipher.doFinal(result);
System.out.println("Decrypted data: " + new String(original));
```

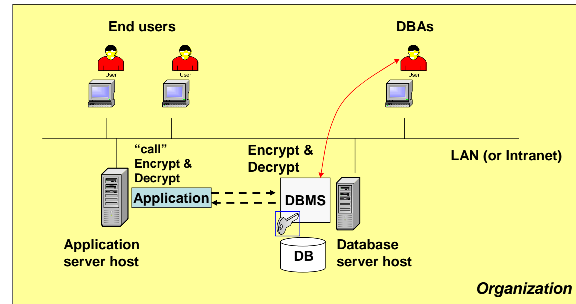
See the Javax.crypto package at:
http://java.sun.com/j2se/1.4.2/docs/api/javax/crypto/package-summary.html

CASE1 - issues

What may happen?

- As more applications need access to encrypted data, the key is duplicated in those applications
- So, the number of people which know the key may become very large ….
- An attacker can easily extract the key from the code…
- Moreover, what happens if the organization decide to change the key? → *find all applications using the key and modify them….*

CASE2 – Encryption/decryption are called by the DB application



CASE2

- The application encrypts/decrypts data using a symmetric key created (and stored by) the DBMS
- How to create a symmetric key in the DBMS (SQL server):
  CREATE SYMMETRIC KEY (Transact-SQL statement)
  CREATE SYMMETRIC KEY SSN_Key_01 WITH ALGORITHM = AES_256 ENCRYPTION BY CERTIFICATE HeatlhC;
- The previous example creates a symmetric key called SSN_Key_01 by using the AES 256 algorithm, and then encrypts the new key with the key in certificate HeatlhC.
- NOTE: in this example the DBMS protects the symmetric key by encrypting it using the key contained in a certificate

CASE2

- The application must obtain the key from the DBMS before using it:
- OPEN SYMMETRIC KEY (Decrypts and loads the key into memory)
- TRANSACT-SQL statement:
  - OPEN SYMMETRIC KEY Key_name DECRYPTION BY <decryption_mechanism>
  - *Key_name* Is the name of the symmetric key to be opened
  - *Decryption mechanism* is the mechanism used to encrypt the symmetric key
- Example:
  - OPEN SYMMETRIC KEY SSN_Key_01 DECRYPTION BY CERTIFICATE HeatlhC;

PURDUE UNIVERSITY

CASE2
the application encrypts the data

```
USE trialdb
GO
-- Create a column in which to store the encrypted data.
ALTER TABLE HumanResources.Employee
   ADD EncryptedNationalIDNumber varbinary(128);  SEE NOTE1 on next slide
GO
-- Open the symmetric key with which to encrypt the data.
OPEN SYMMETRIC KEY SSN_Key_01
  DECRYPTION BY CERTIFICATE HeatlhC;
-- Encrypt the value in column NationalIDNumber with symmetric key
-- SSN_Key_01. Save the result in column EncryptedNationalIDNumber.
UPDATE HumanResources.Employee
SET EncryptedNationalIDNumber
   = EncryptByKey(Key_GUID('SSN_Key_01'), NationalIDNumber); SEE NOTE1 on next
     slide
---------------------------
```

The Key_GUID function returns the GUID of a symmetric key in the database. The GUID serves as an identifier for the key and it is stored in metadata (SELECT key_guid FROM sys.symmetric_keys). It is used for finding the corresponding key

PURDUE UNIVERSITY

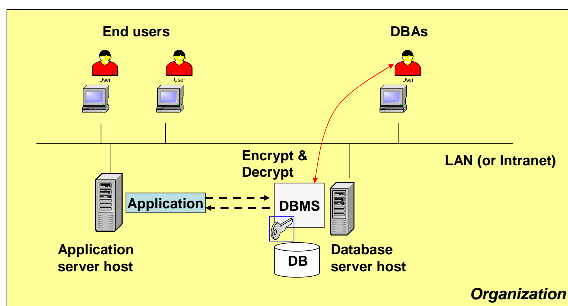
CASE2
the application encrypts the data

NOTE1

- The symmetric key encryption functions all return varbinary data with a maximum size of 8,000 bytes.
- The Decrypt functions return up to 8,000 bytes of clear text varbinary data from encrypted cipher text, which also limits the amount of data you can encrypt without breaking it into chunks.
- Since the Decrypt functions also return varbinary data, it is necessary to cast the decrypted data back to the original data type for use.

PURDUE UNIVERSITY

CASE3 – Encryption/decryption are transparent to the BD application



PURDUE UNIVERSITY

CASE3 SQL Server – Transparent Data Encryption

- Transparent Data Encryption (TDE) allows users to encrypt the sensitive data in the database and protect the keys that are used to encrypt the data with a certificate.
- TDE performs real-time I/O encryption and decryption of the data and log files.
- The encryption uses a database encryption key (DEK), which is stored in the database boot record for availability during recovery.
- The DEK is a symmetric key secured by using a certificate stored in the master database of the server
- Encryption of the database file is performed at the page level. The pages in an encrypted database are encrypted before they are written to disk and decrypted when read into memory.

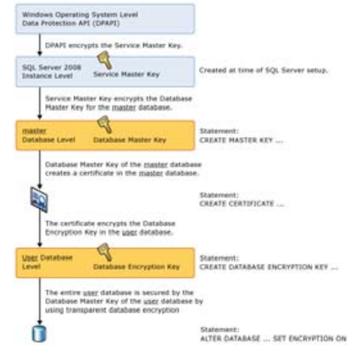
PURDUE UNIVERSITY

CASE3 SQL Server – Transparent Data Encryption

- With TDE, the user DB application does not have to encrypt/derypt data by itself
- With TDE, all the database tables are encrypted

PURDUE UNIVERSITY

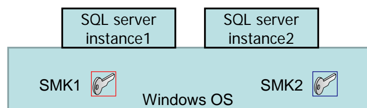
CASE3 SQL Server – TDE Architecture



PURDUE UNIVERSITY

TDE – encryption key hierarchy

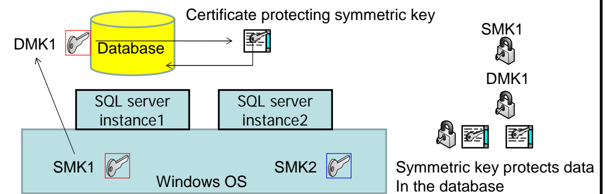
- Purpose: to organize encryption keys in a cryptography hierarchy
- A Service Master Key (SKM) is associated with each DB server instance. This SMK is protected by the Windows OS via Windows Data Protection Api



PURDUE UNIVERSITY

TDE – encryption key hierarchy

- The SMK protects the database master key (DMK), which is stored at the user database level and which in turn protects certificates and asymmetric keys.
- These in turn protect symmetric keys, which protect the data.



PURDUE UNIVERSITY

Example

The following example illustrates encrypting the AdventureWorks database using a certificate installed on the server named MyServerCert.

```
USE master;
GO
CREATE MASTER KEY ENCRYPTION BY PASSWORD = '<UseStrongPasswordHere>';
go
CREATE CERTIFICATE MyServerCert WITH SUBJECT = 'My DEK Certificate'
go
USE AdventureWorks
GO
CREATE DATABASE ENCRYPTION KEY
WITH ALGORITHM = AES_128
ENCRYPTION BY SERVER CERTIFICATE MyServerCert
GO
ALTER DATABASE AdventureWorks
SET ENCRYPTION ON
GO
```

CREATE MASTER KEY ENCRYPTION BY PASSWORD creates an asymmetric key;
This key is encrypted by using triple DES and the user-supplied password
Note1: data are encrypted symmetric keys for performance reasons
Note2: to enable TDE, a user must have the permission of creating a DMK and the CONTROL authorization on the user database

PURDUE UNIVERSITY

What is Encrypted

- TDE operates at the I/O level through the buffer pool. Thus any data that is written into the database file is encrypted
- Snapshots and backups are encrypted
- The transaction log is also encrypted (but additional caveats apply)
- Data that is in use, however, is not encrypted because TDE does not provide protection at the memory or transit level

PURDUE UNIVERSITY

Comparison with cell-level encryption

- Cell-level encryption has some advantages over DB-level encryption:
  - If offers a more granular level of encryption; one needs only to encrypt the data that are sensitive
  - Data is not decrypted until it is used so that even if a page is loaded in memory, sensitive data is not in clear text
  - It supports explicit key management; users can have their own keys for their own data
- And some disadvantages:
  - Applications have to be changed
  - The domains of columns storing encrypted data need to be changed to **varbinary**
  - Performance is affected because indexes on encrypted columns offer no advantage so equality and range queries result in full table scans. The performance of a basic query (selects and decrypts a single encrypted column) tends to be around 20% worse (3-5% on average for TDE)

PURDUE UNIVERSITY

Additional Considerations

TDE and cell-level encryption accomplish two different objectives:

- if the amount of data that must be encrypted is very small or if the application can be custom designed to use it and performance is not a concern, cell-level encryption is to be preferred
- Otherwise, TDE is to be preferred

PURDUE UNIVERSITY

Documentation

- SQL server encryption http://msdn.microsoft.com/en-us/library/cc278098.aspx
- Transparent Data encryption http://msdn.microsoft.com/en-us/library/bb934049.aspx
- CREATE SYMMETRIC KEY (Transact-SQL) http://msdn.microsoft.com/en-us/library/ms188357.aspx
- ENCRYPTBYKEY (Transact-SQL) http://msdn.microsoft.com/en-us/library/ms174361.aspx
- DECRYPTBYKEY (Transact-SQL) http://msdn.microsoft.com/en-us/library/ms181860.aspx