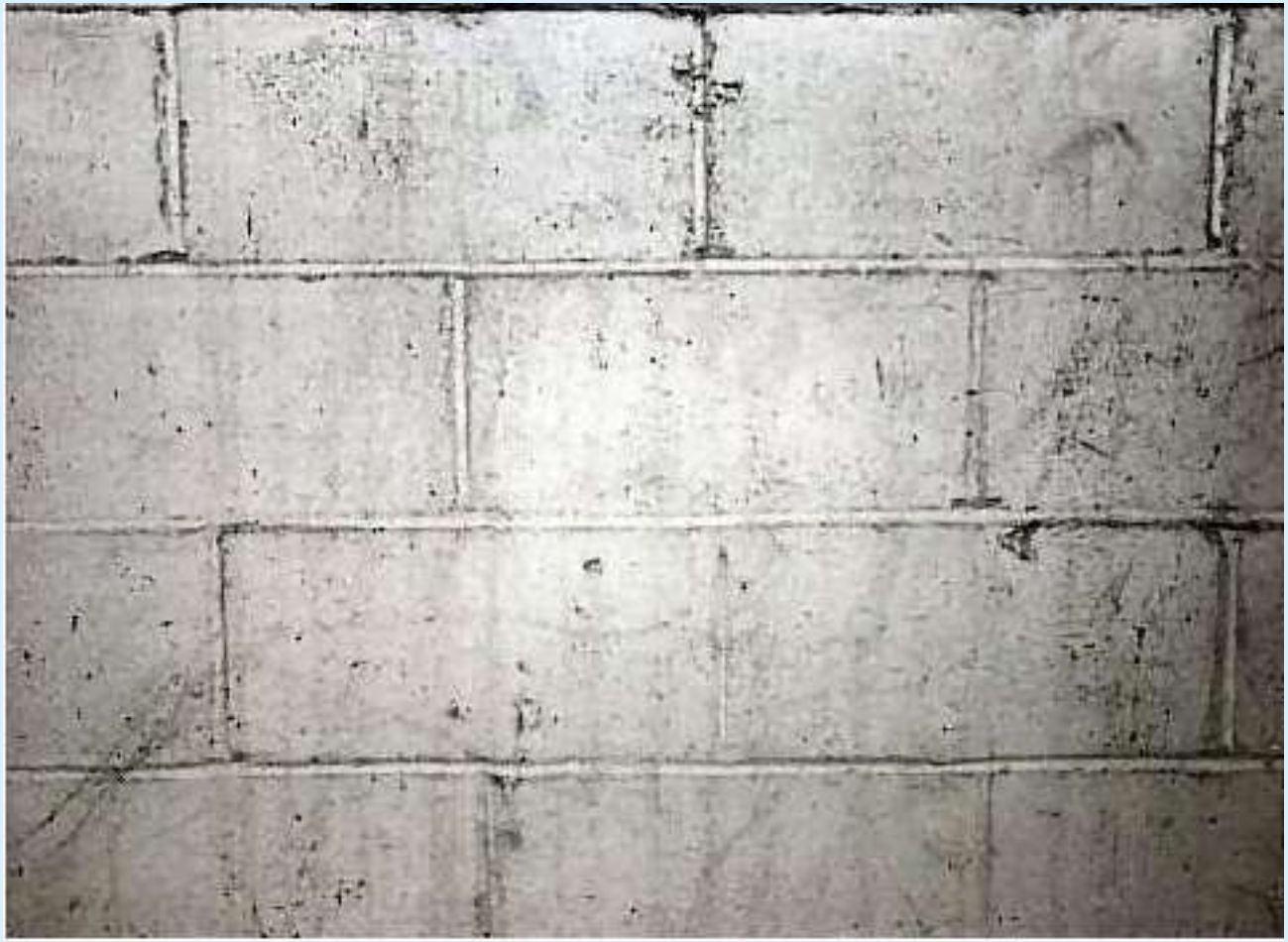
DATABASE ENCRYPTION

By: Samer Hamzeh
Hussein Ashmar

A Truly Secure Database



Encryption

Algorithms

An algorithm is basically a procedure or a formula for solving a data snooping problem. An encryption algorithm is a set of mathematical procedure for performing encryption on data. Through the use of such an algorithm, information is made in the cipher text and requires the use of a key to transforming the data into its original form. This brings us to the concept of cryptography that has long been used in information security in communication systems.

Cryptography

Cryptography is a method of using advanced mathematical principles in storing and transmitting data in a particular form so that only those whom it is intended can read and process it. Encryption is a key concept in cryptography – It is a process whereby a message is encoded in a format that cannot be read or understood by an eavesdropper. The technique is old and was first used by Caesar to encrypt his messages using Caesar cipher. A plain text from a user can be encrypted to a ciphertext, then send through a communication channel and no eavesdropper can interfere with the plain text. When it reaches the receiver end, the ciphertext is decrypted to the original plain text.

Encryption

Cryptography Terms:

Encryption: It is the process of locking up information using cryptography. Information that has been locked this way is encrypted.

Decryption: The process of unlocking the encrypted information using cryptographic techniques.

Key: A secret like a password used to encrypt and decrypt information. There are a few different types of keys used in cryptography.

Steganography: It is actually the science of hiding information from people who would snoop on you. The difference between steganography and encryption is that the would-be snoopers may not be able to tell there's any hidden information in the first place.

Threats

External Threats

- ✓ Hackers breach a software company's website, stealing credit card information.

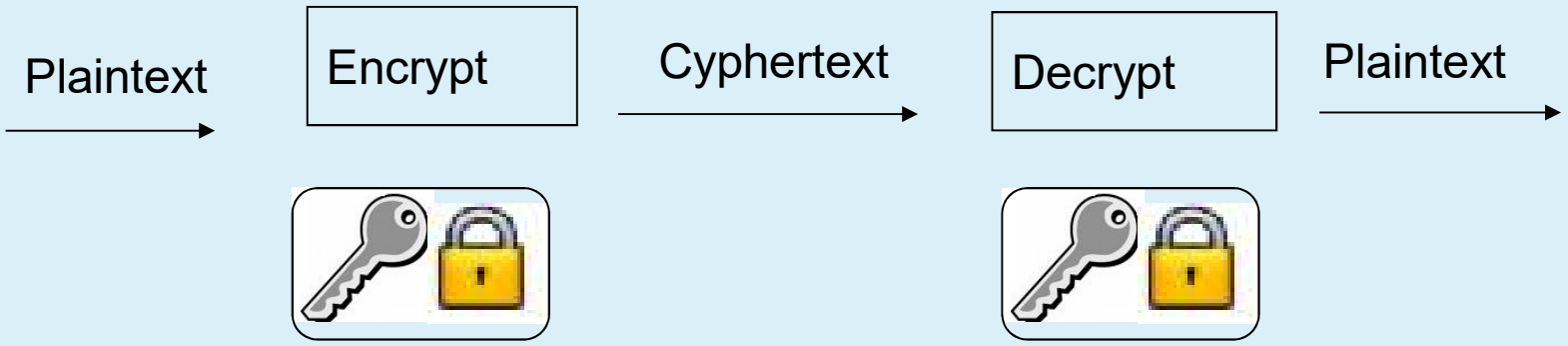
Internal Threats

- ✓ A disgruntled employee accesses confidential salary information and distributes it.

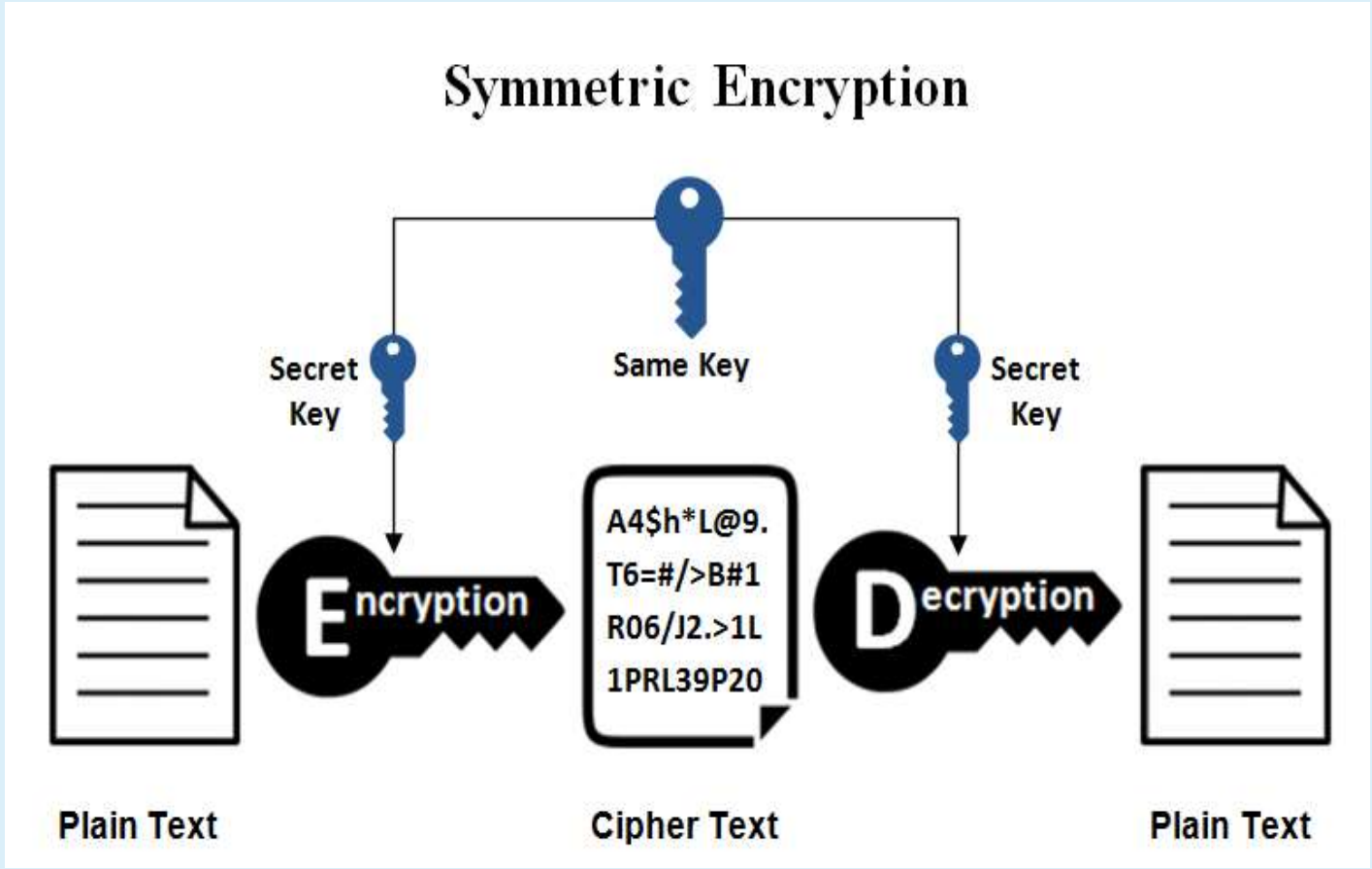
Physical threats

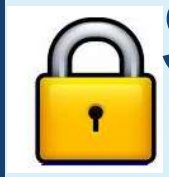
- ✓ Thieves strike a data center.

The Encryption Process



Symmetric key cryptography





Symmetric key cryptography

- Algorithm: DES, 3DES AND AES
- Faster than asymmetric cryptography
- PROs

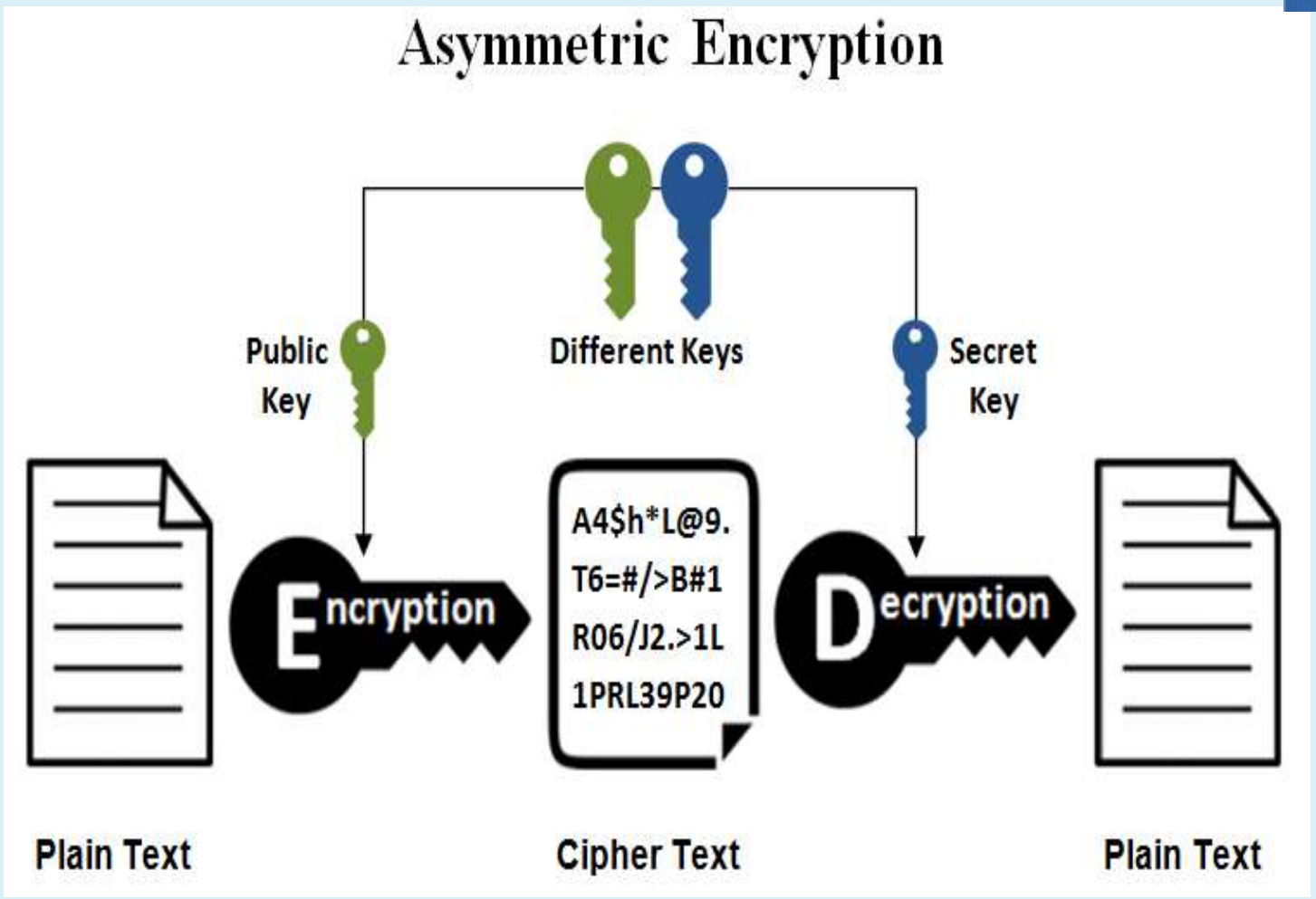
- Performance

- 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


Asymmetric Encryption
Public Key
Different Keys
Secret Key
Encryption
A4$h*L@9.
T6=#/>B#1
R06/J2.>1L
1PRL39P20
Decryption
Plain Text
Cipher Text
Plain Text

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
- Encryption → Public Key
- Decryption →Secret Private Key
- Difference Between Symmetric and Asymmetric Encryption
- Symmetric encryption uses a single key that needs to be shared among the people who need to receive the message while asymmetric encryption uses a pair of public key and a private key to encrypt and decrypt messages when communicating.
- Symmetric encryption is an old technique while asymmetric encryption is relatively new.
- Asymmetric encryption was introduced to complement the inherent problem of the need to share the key in symmetric encryption model, eliminating the need to share the key by using a pair of public-private keys.
- Asymmetric encryption takes relatively more time than the symmetric encryption.

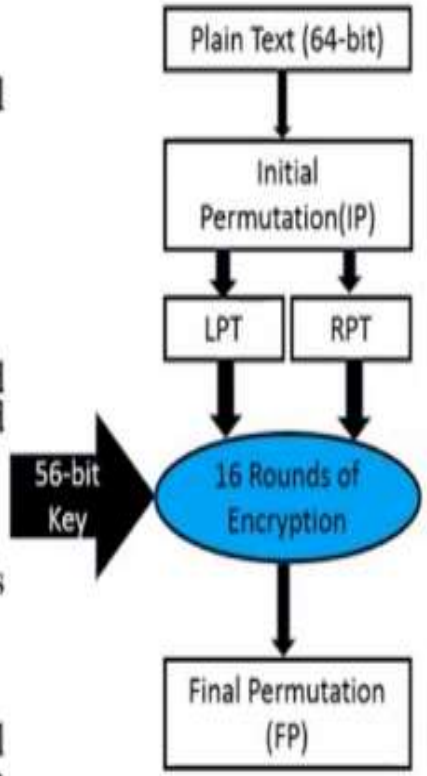
Encryption Algorithms: Data Encryption Standard

- DES has a short (56 bit) key plus 8 bits used for parity checking
- Very susceptible to brute force attacks
- Now outdated – older versions of DBMS encryption routines used DES e.g. early versions of Oracle

DES (Data Encryption Standard)

❖ Steps of DES

1. 64-bit plain text block is given to Initial Permutation (IP) function.
2. IP performed on 64-bit plain text block.
3. IP produced two halves of the permuted block known as Left Plain Text (LPT) and Right Plain Text (RPT).
4. Each LPT and RPT performed 16-rounds of encryption process.
5. LPT and RPT rejoined and Final Permutation (FP) is performed on combined block.



DES Block Cipher - Key Creation:

1. 64bit Key --> PC - 1 --> 56bit KeyP
2. Divide Kp into L, R
3. Rotate shift
4. Concatenate L + R
5. Key[i] --> PC-2

The last bit of each byte is used as a parity bit.

57	49	41	33	25	17	9
1	58	50	42	34	26	18
10	2	59	51	43	35	27
19	11	3	60	52	44	36
63	55	47	39	31	23	15
7	62	54	46	38	30	22
14	6	61	53	45	37	29
21	13	5	28	20	12	4

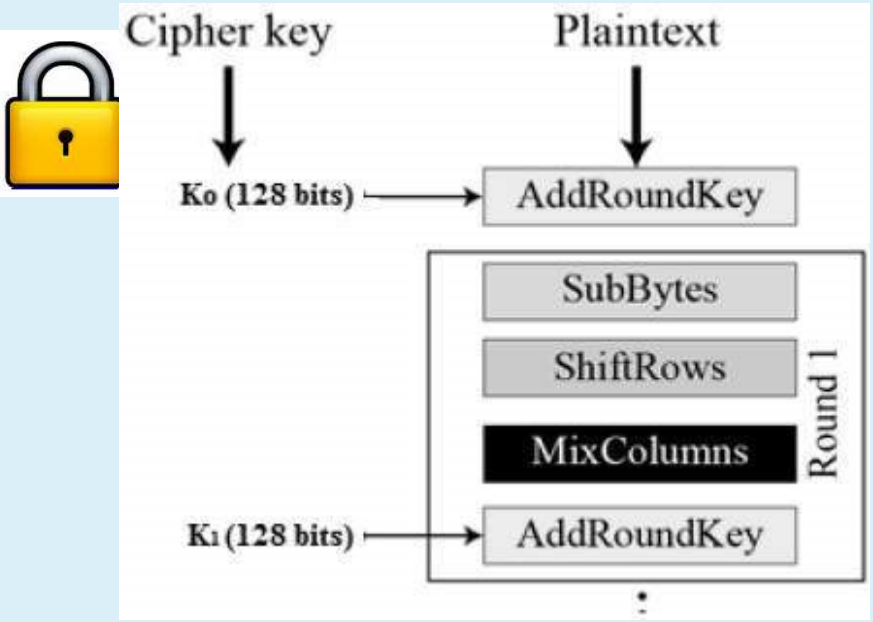
Permuted Choice -1

K= 00010011 00110100 01010111 01111001 10011011 10111100 11011111 11110001

K_p=1111000 0110011 0010101 0101111 0101010 1011001 1001111 0001111

Encryption Algorithms: Advanced Encryption Standard AES

- Key size 128,192 or 256 bits
- Consists of a set of processing rounds – the number varies depending on the key size e.g. 14 rounds for 256 length keys
- More secure



Block to state - example

Text	A	E	S	U	S	E	S	A	M	A	T	R	I	X	Z	Z
Hexadecimal	00	04	12	14	12	04	12	00	0C	00	13	11	08	23	19	19

$$\begin{bmatrix} 00 & 12 & 0C & 08 \\ 04 & 04 & 00 & 23 \\ 12 & 12 & 13 & 19 \\ 14 & 00 & 11 & 19 \end{bmatrix}$$ State

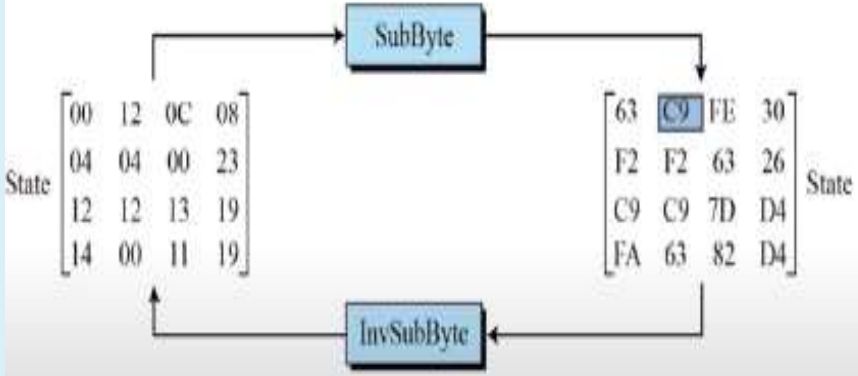
SubByte

- During encryption each value of the state is replaced with the corresponding SBOX value
- For example HEX 19 would get replaced with HEX D4

AES S-Box Lookup Table

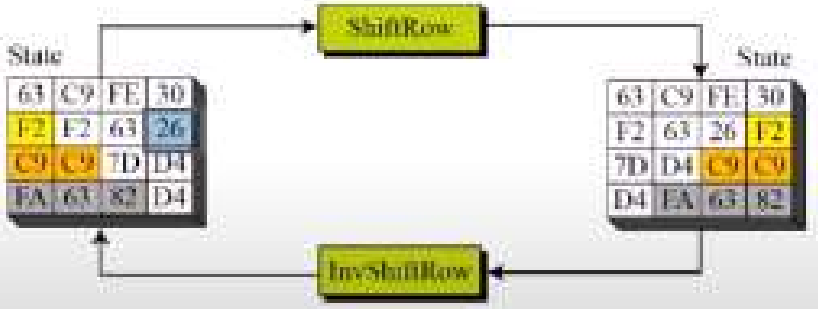
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
6	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
B	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
D	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

SubByte Example



ShiftRows

- Each row is moved over (shifted) 1, 2 or 3 spaces over to the right, depending on the row of the state.
- First row is never shifted Row1 0 Row2 1 Row3 2 Roe4 3



MixColumns

- The MixColumns transformation operates at the column level; it transforms each column of the state to a new column.

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \quad \text{Inverse} \quad \begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix}$$

$C \qquad C^{-1}$

MixColumns

- The first result byte is calculated by multiplying 4 values of the state column against 4 values of the first row of the matrix. The result of each multiplication is then XORed to produce 1 Byte.
  - b1 = (b1 * 2) XOR (b2*3) XOR (b3*1) XOR (b4*1)
- The second result byte is calculated by multiplying the same 4 values of the state column against 4 values of the second row of the matrix. The result of each multiplication is then XORed to produce 1 Byte.
  - b2 = (b1 * 1) XOR (b2*2) XOR (b3*3) XOR (b4*1)
- The third result byte is calculated by multiplying the same 4 values of the state column against 4 values of the third row of the matrix. The result of each multiplication is then XORed to produce 1 Byte.
  - b3 = (b1 * 1) XOR (b2*1) XOR (b3*2) XOR (b4*3)
- The fourth result byte is calculated by multiplying the same 4 values of the state column against 4 values of the fourth row of the matrix. The result of each multiplication is then XORed to produce 1 Byte.
  - b4 = (b1 * 3) XOR (b2*1) XOR (b3*1) XOR (b4*2)

Multiplication Matrix

2	3	1	1
1	2	3	1
1	1	2	3
3	1	1	2

16 byte State

b1	b5	b9	b13
b2	b6	b10	b14
b3	b7	b11	b15
b4	b8	b12	b16

What is the acceptable performance hit?

- Most shops avoid encrypting data because of fear of how it will impact performance and code.

Take a sample Query from database:

```
SELECT c.companyname, c.contactname, c.address, c.city, c.country,

o.orderdate, o.requireddate, o.shipaddress, o.shipcity, o.shipcountry

FROM Orders o

JOIN Customers c ON (o.custid = c.custid)

WHERE o.shipcountry = 'USA'
```

Before TDE is enabled, CPU utilization averaged **22%** utilization for the duration of the test. Query elapsed time was about **48 ms.**

After TDE is enabled using the steps above and ran the same query as before. CPU utilization averaged **28%** utilization for the duration of the test. Query elapsed time was about **253 ms.**

Before Deciding on Encryption

- Know the data and the database
  - *What should be encrypted?*
  - *Which encryption algorithms?*
  - *DBMS or external encryption?*
  - *What is the acceptable performance hit?*
  - *Who are you protecting against?*
  - *Is the benefit worth the cost?*

What should be encrypted?

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

Partial Data Encryption

- Partial encryption provides more granularity plus the data is not decrypted until it is used
- Usually referring to column encryption although it can also be cell level or encryption of DB objects such as triggers
- Rule of thumb – encrypting a single column is likely to produce a 5% performance hit, but this varies wildly

Where should encryption/ decryption be performed?

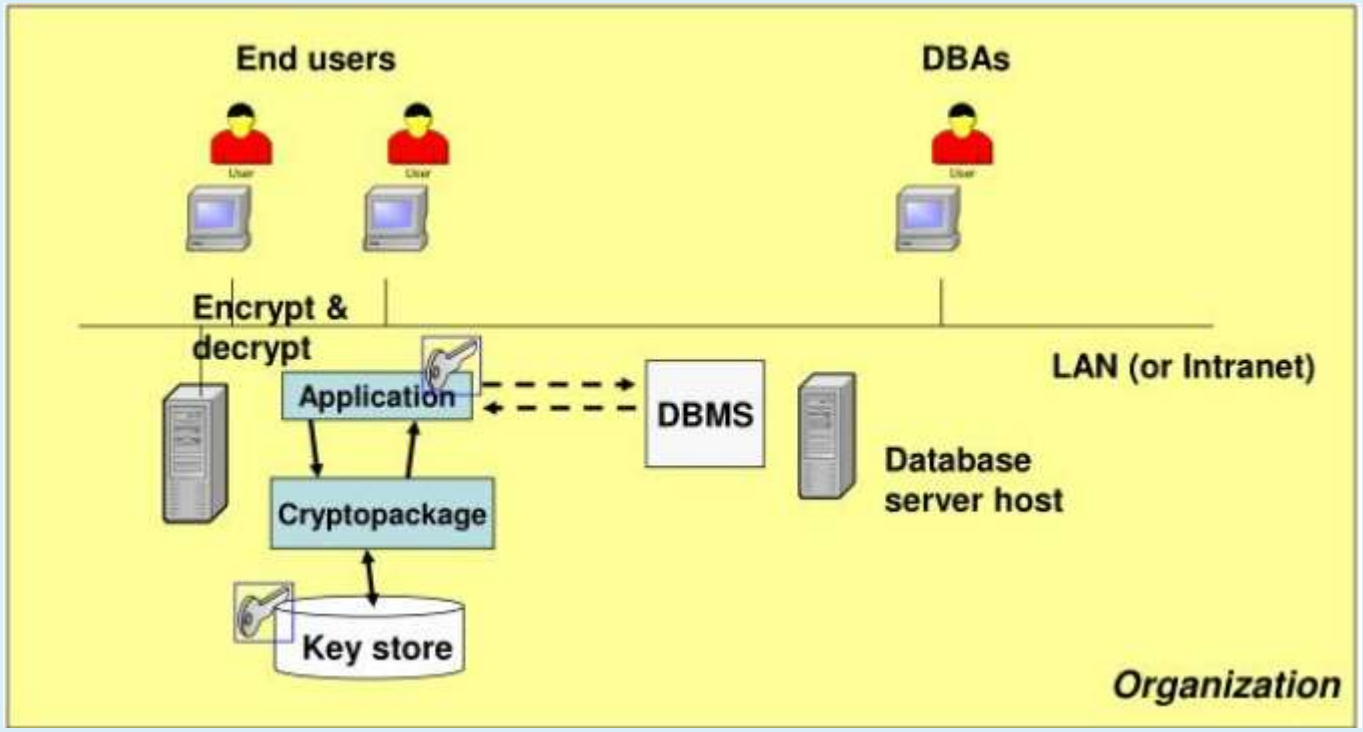
- The application must request encryption and decryption.

• CASE1 - In this case, keys are managed outside the DBMS (i.e. encrypt/decryption performed by some package)

• CASE2 - In this case, keys are managed by the DBMS (i.e. encrypt/decryption 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”

Case1-Application-level encryption



Case1-Application-level encryption

- 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

Case1-Application-level encryption

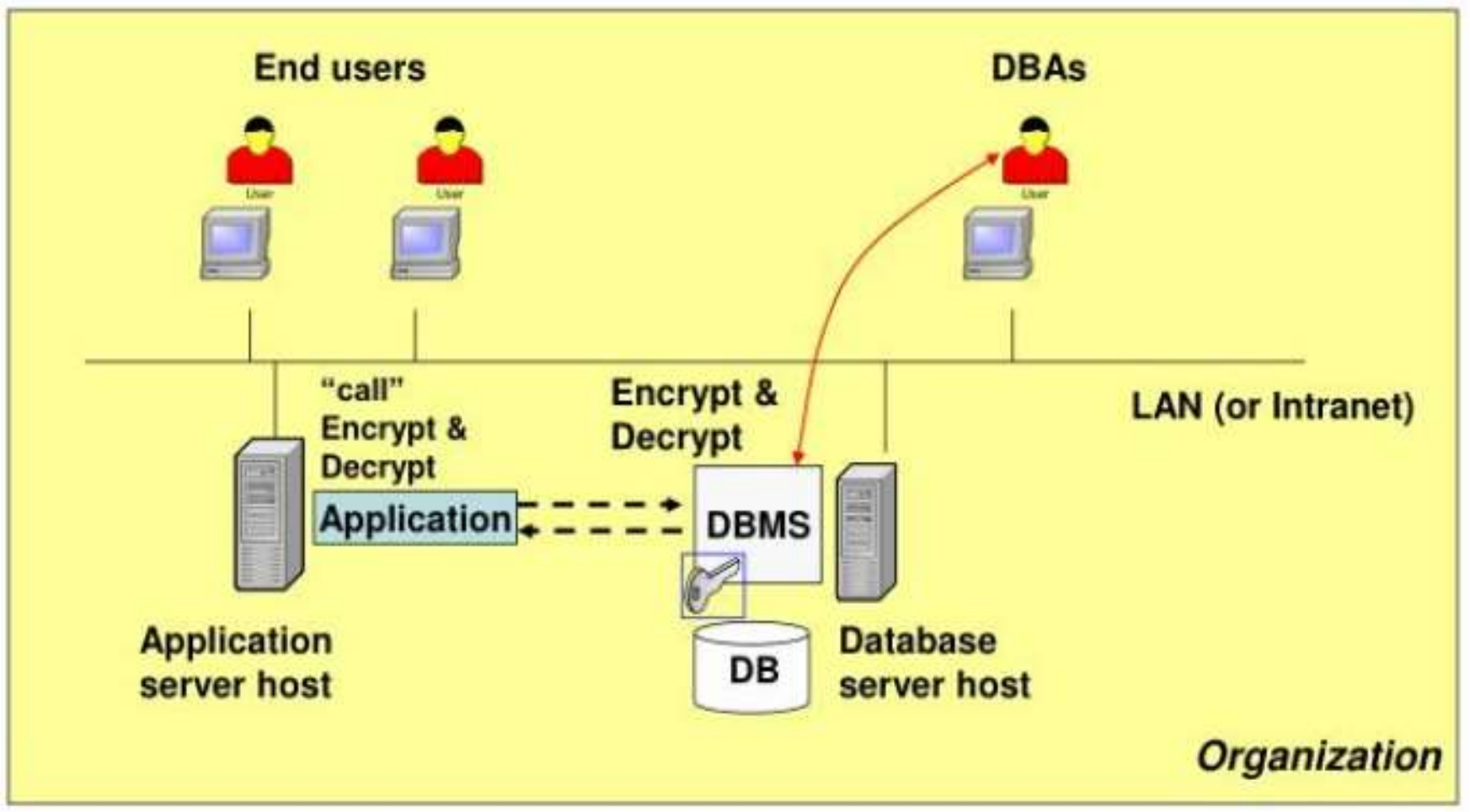
The application encrypts data before inserting them in the DB. Schematically:

- 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

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 – Encryption /decryption are called by the DB

- 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.
- The DBMS protects the symmetric key by encrypting it using the key contained in a certificate

CASE2 the application encrypts the data

- 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;

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

CASE2 the application encrypts the data

- NOTE1

• The symmetric key encryption functions all return varbinary data with a maximum size of 8,000 bytes.

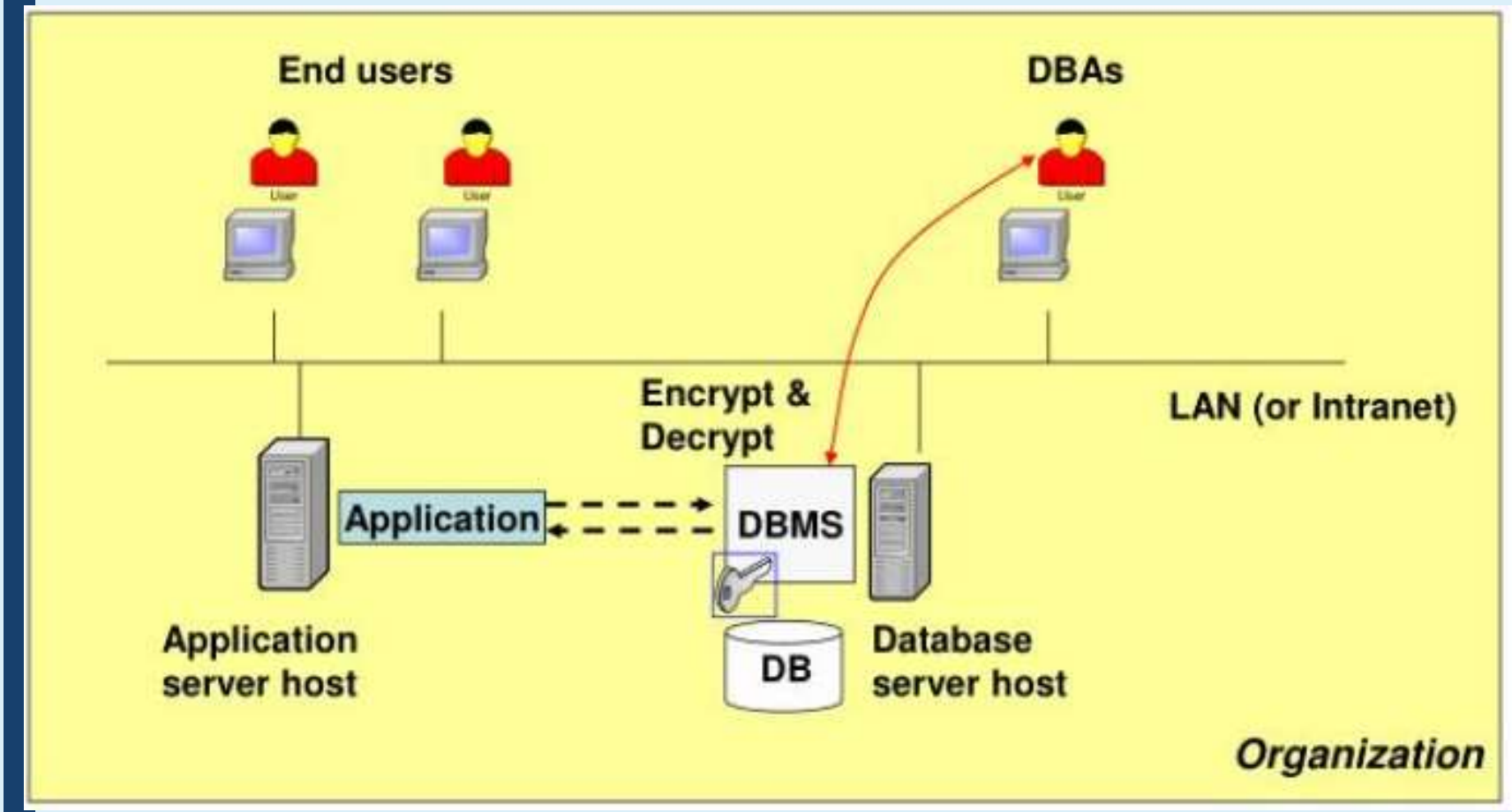
• The Decrypt functions return up to 8,000 bytes of clear text varbinary data from encrypted cipher text, which also limits the amount of data you can encrypt without breaking it into chunks.

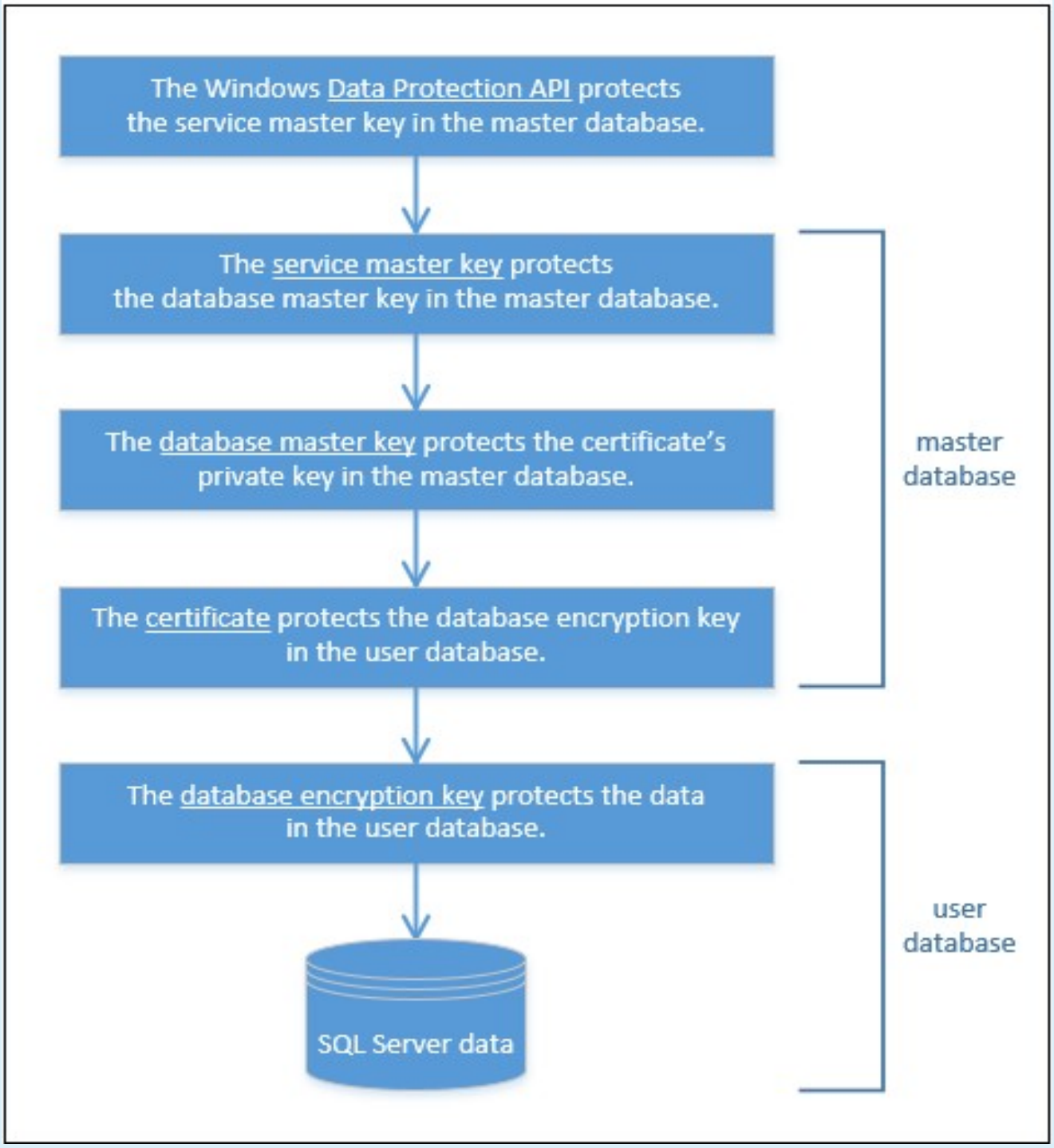
• Since the Decrypt functions also return varbinary data, it is necessary to cast the decrypted data back to the original data type for use.

Whole Database Encryption

- The whole database is encrypted
- This protects the data at rest but requires decryption for use
- Whole DB encryption has traditionally been regarded as too expensive – SQL Server TDE, new with 2008, claims to reduce the performance hit but still acknowledges a cost (1)

CASE3 – Encryption / decryption are transparent to the BD




The Windows Data Protection API protects the service master key in the master database.
The service master key protects the database master key in the master database.
The database master key protects the certificate's private key in the master database.
The certificate protects the database encryption key in the user database.
The database encryption key protects the data in the user database.
SQL Server data
master database
user database

CASE3 – Encryption/decryption are transparent to the DB

- Transparent data encryption (TDE) executes encryption and decryption within the database engine itself. This method doesn't require code modification of the database or application and is easier for admins to manage. Since it's a particularly popular method of database encryption.

CASE3 – Encryption/decryption are transparent to the BD

- This type of encryption is “transparent” because it is invisible to users and applications that are drawing on the data and is easily used without making any application-level changes. It is decrypted for authorized users or applications when in use but remains protected at rest. Even if the physical media is compromised or the files stolen, the data as a whole remains unreadable—only authorized users can successfully read the data
- This provides a disincentive for hackers to steal the data at all. When all is said and done, using TDE can help a business remain in compliance with a range of specific security regulations.

CASE3 – Encryption/decryption are transparent to the BD

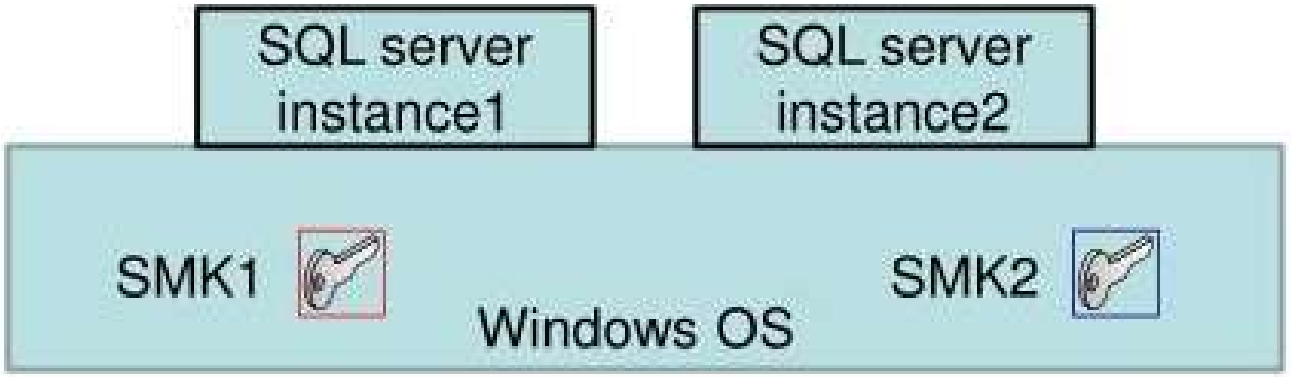
- TDE performs real-time I/O encryption and decryption of the data and log files. The encryption uses a database encryption key (DEK), which is stored in the database boot record for availability during recovery.
- The DEK is a symmetric key secured by using a certificate stored in the master database 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.

CASE3 – Encryption/decryption are transparent to the BD

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

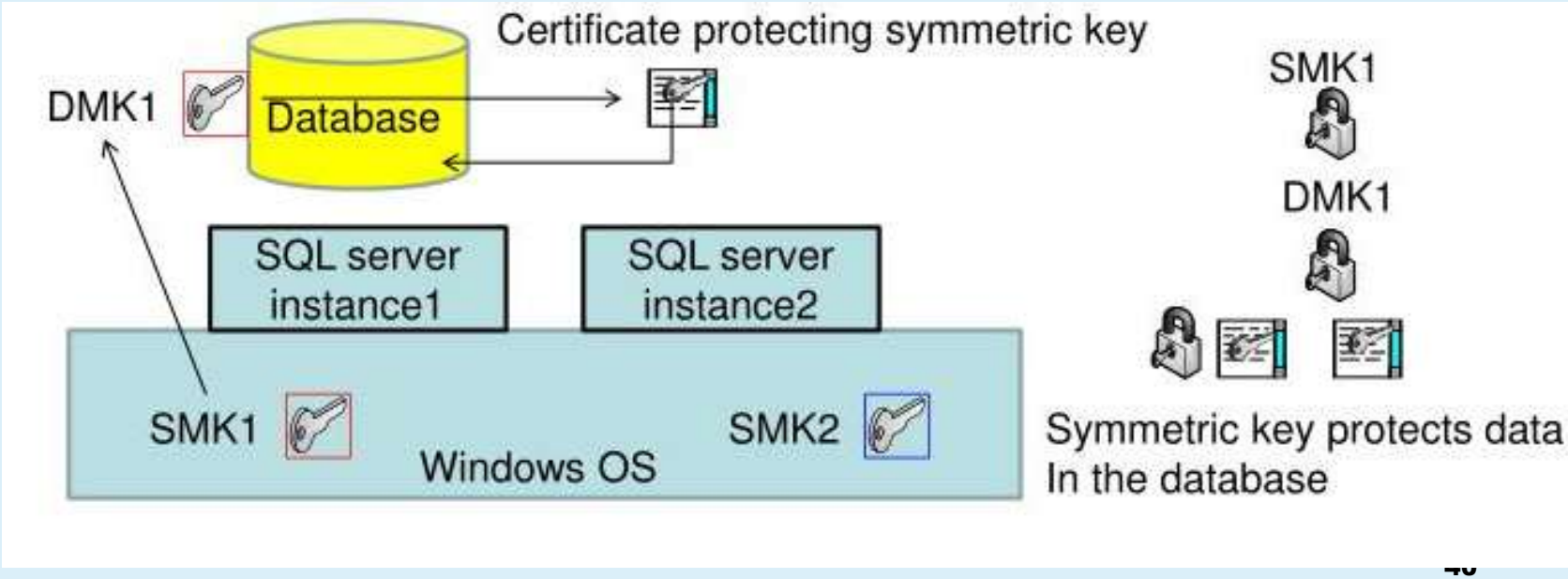
TDE – encryption key hierarchy

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



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 symmetric keys. These in turn protect symmetric keys, which protect the data.



Example

```
USE master;
GO
CREATE MASTER KEY ENCRYPTION BY PASSWORD = '<UseStrongPasswordHere>';
go
CREATE CERTIFICATE MyServerCert WITH SUBJECT = 'My DEK Certificate';
go
USE AdventureWorks2012;
GO
CREATE DATABASE ENCRYPTION KEY
WITH ALGORITHM = AES_128
ENCRYPTION BY SERVER CERTIFICATE MyServerCert;
GO
ALTER DATABASE AdventureWorks2012
SET ENCRYPTION ON;
GO
```

CREATE MASTER KEY ENCRYPTION BY PASSWORD creates an symmetric key; This key is encrypted by using triple DES and the user-supplied password

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*

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

References

1. https://www.ssl2buy.com/wiki/symmetric-vs-asymmetric-encryption-what-are-differences#:~:text=Symmetric%20encryption%20uses%20a%20single,and%20decrypt%20messages%20when%20communicating.

2. https://docs.microsoft.com/en-us/sql/relational-databases/security/encryption/transparent-data-encryption?redirectedfrom=MSDN&view=sql-server-ver15

3. https://docs.microsoft.com/en-us/sql/t-sql/statements/create-symmetric-key-transact-sql?redirectedfrom=MSDN&view=sql-server-ver15

4. http://msdn.microsoft.com/en-us/library/cc278098.aspx

5. http://msdn.microsoft.com/en-us/library/bb934049.aspx

6. http://www.tropsoft.com/strongenc/des3.htm