Database Security

CT069-3-3

Encryption

Learning Outcomes

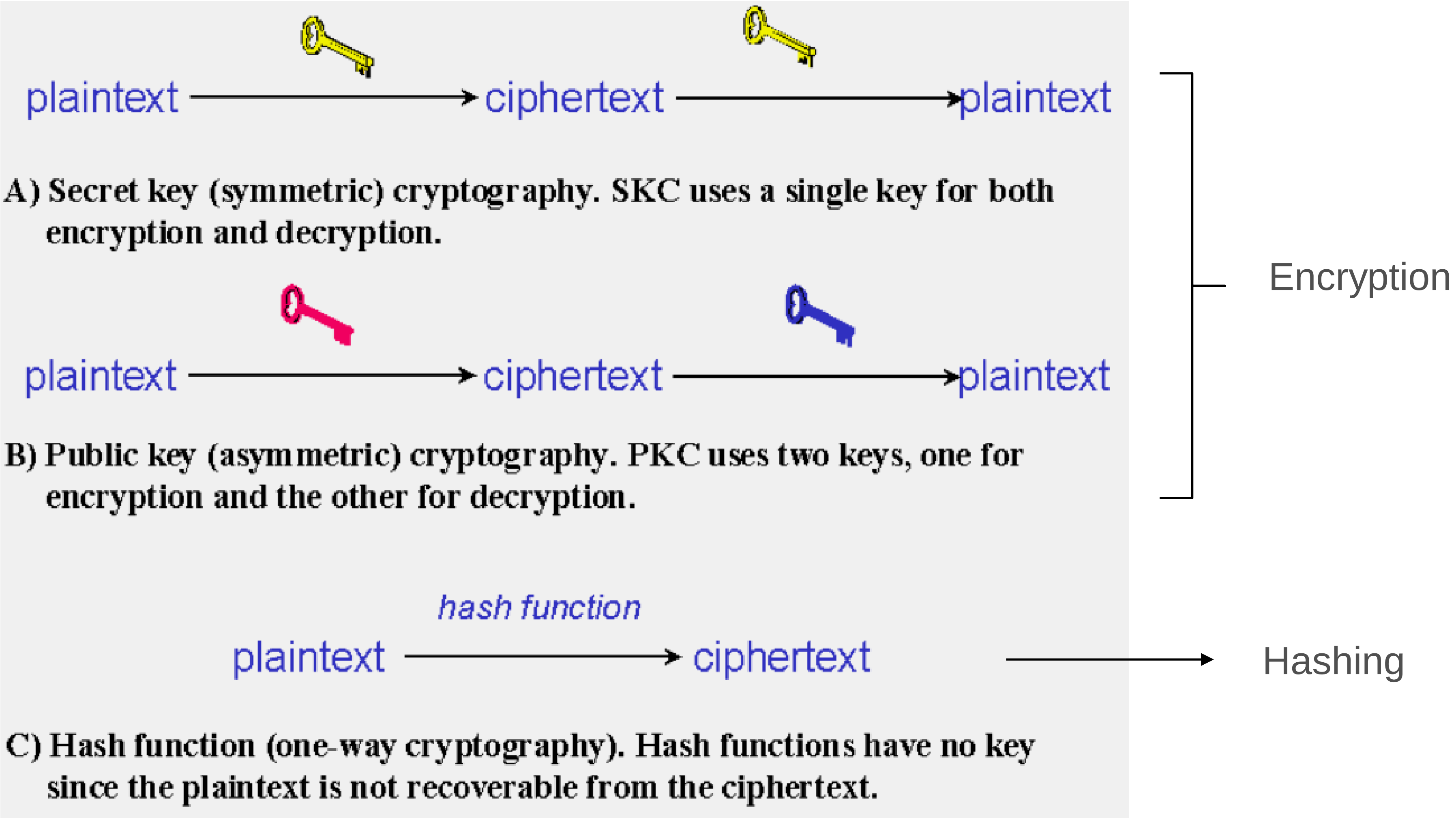
At the end of this lecture, YOU should be able to :

- Explain the concepts and principles of secure database using encryption and hashing
- Implement database and column level encryptions using keys and/certificates.

Cryptography

- Cryptography is a science of secret writing
- Cryptography is associated with the process of converting ordinary plain text into unintelligible text and vice-versa.
- It is a method of storing and transmitting data in a particular form so that only those for whom it is intended can read and process it.
- Cryptography not only protects data from theft or alteration but can also be used for user authentication.

Cryptography



A) Secret key (symmetric) cryptography. SKC uses a single key for both encryption and decryption.

B) Public key (asymmetric) cryptography. PKC uses two keys, one for encryption and the other for decryption.

C) Hash function (one-way cryptography). Hash functions have no key since the plaintext is not recoverable from the ciphertext.

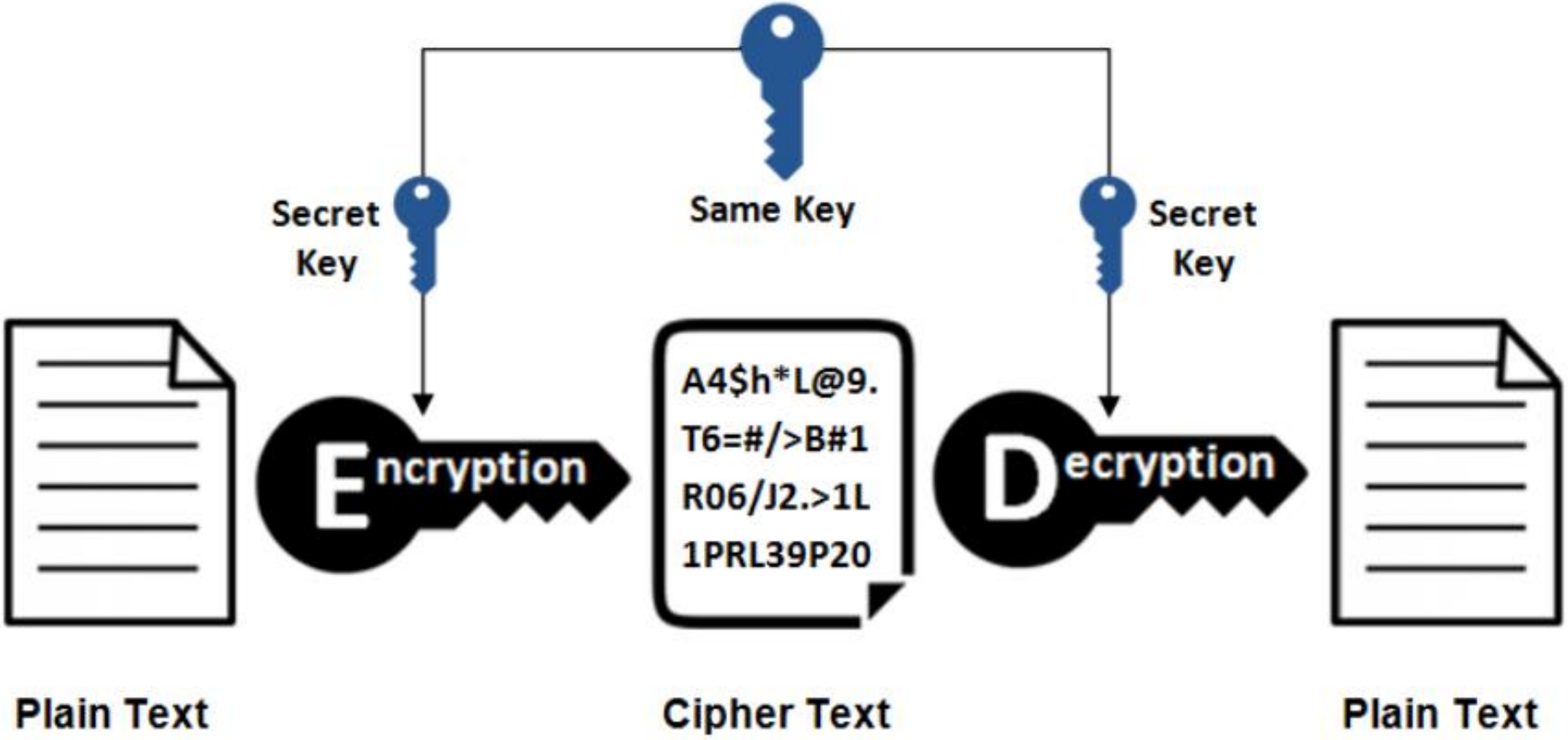
Encryption

Hashing

Encryption

- **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.
- Encryption **Key**: A secret like a password used to encrypt and decrypt information. There are a few different types of keys used in cryptography.
- Encryption **Certificate**: A digital file to perform encryption.

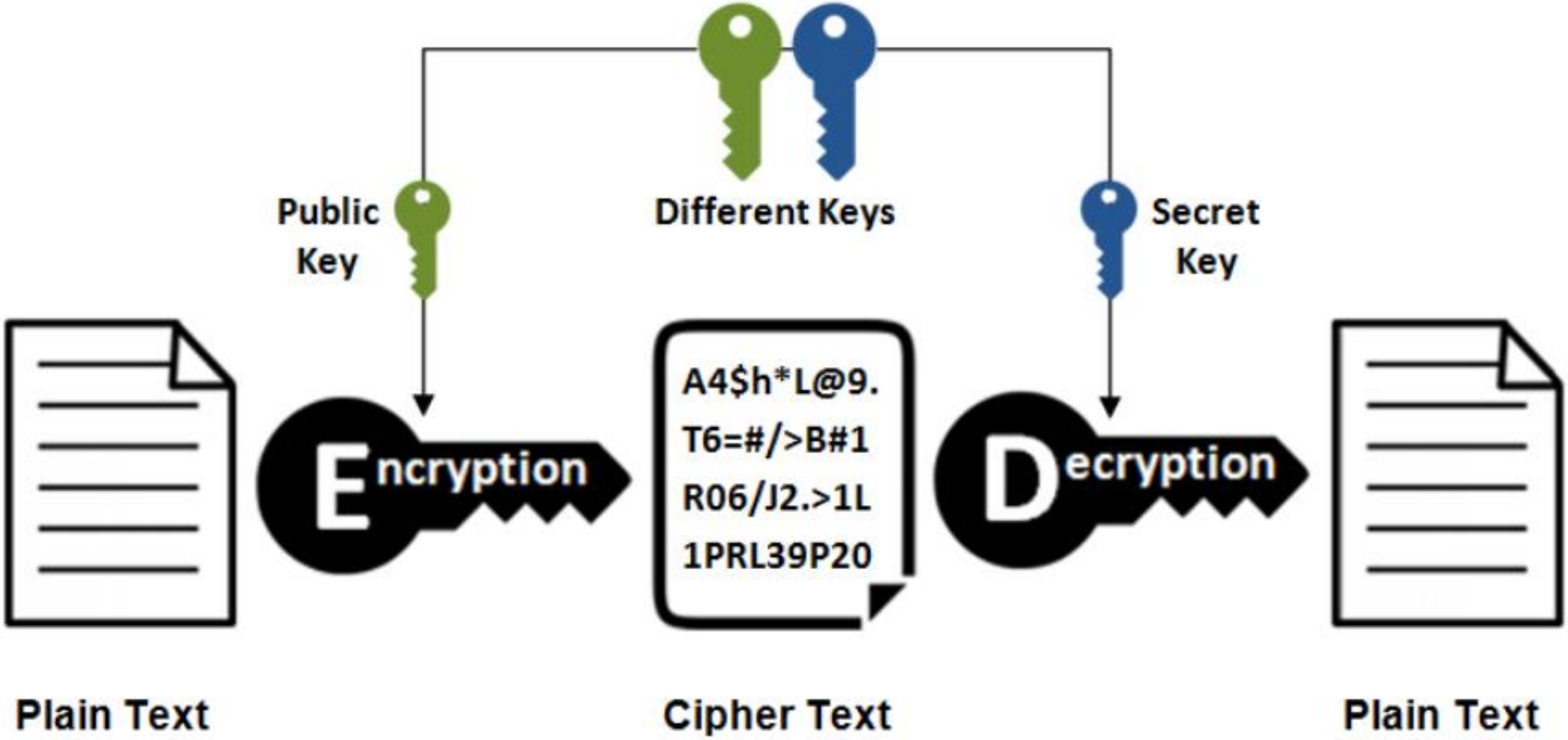
Symmetric Encryption



Symmetric Encryption

- Also known as Secret Key Cryptography (SKC)
- Primarily used for privacy and confidentiality.
- One secret key to cipher and decipher information.
- Symmetric encryption is an old and best-known technique.
- It uses a secret key that can either be a number, a word or a string of random letters. It is a blended with the plain text of a message to change the content in a particular way.
- The sender and the recipient should know the secret key that is used to encrypt and decrypt all the messages.
- Blowfish, AES, RC4, DES, RC5, and RC6 are examples of symmetric encryption.
- The main disadvantage of the symmetric key encryption is that all parties involved must exchange the key used to encrypt the data before they can decrypt it.

Asymmetric Encryption



Asymmetric Encryption

- Primarily used for authentication, non-repudiation, and key exchange.
- Asymmetric encryption also known as public key cryptography, uses two keys to encrypt & decrypt a plain text.
- A public key is made freely available to anyone who might want to send you a message. The second private key is kept a secret so that you can only know.
- A message that is encrypted using a public key can only be decrypted using a private key. Security of the public key is not required because it is publicly available and can be passed over the internet.
- Asymmetric encryption is mostly used in day-to-day communication channels, especially over the Internet.
- Popular asymmetric key encryption algorithm includes EIGamal, RSA, DSA, Elliptic curve techniques, PKCS.
- An asymmetric key can be used to encrypt a symmetric key for storage in a database.

Symmetric vs Asymmetric

Key Differences	Symmetric Encryption	Asymmetric Encryption
Size of cipher text	Smaller cipher text compares to original plain text file.	Larger cipher text compares to original plain text file.
Resource Utilization	Symmetric key encryption works on low usage of resources.	Asymmetric encryption requires high consumption of resources.
Key Lengths	128 or 256-bit key size.	RSA 2048-bit or higher key size.
Security	Less secured due to use a single key for encryption.	Much safer as two keys are involved in encryption and decryption.
Speed	Symmetric encryption is fast technique	Asymmetric encryption is slower in terms of speed.
Algorithms	RC4, AES, DES, 3DES, and QUAD.	RSA, Diffie-Hellman, ECC algorithms.

Certificates

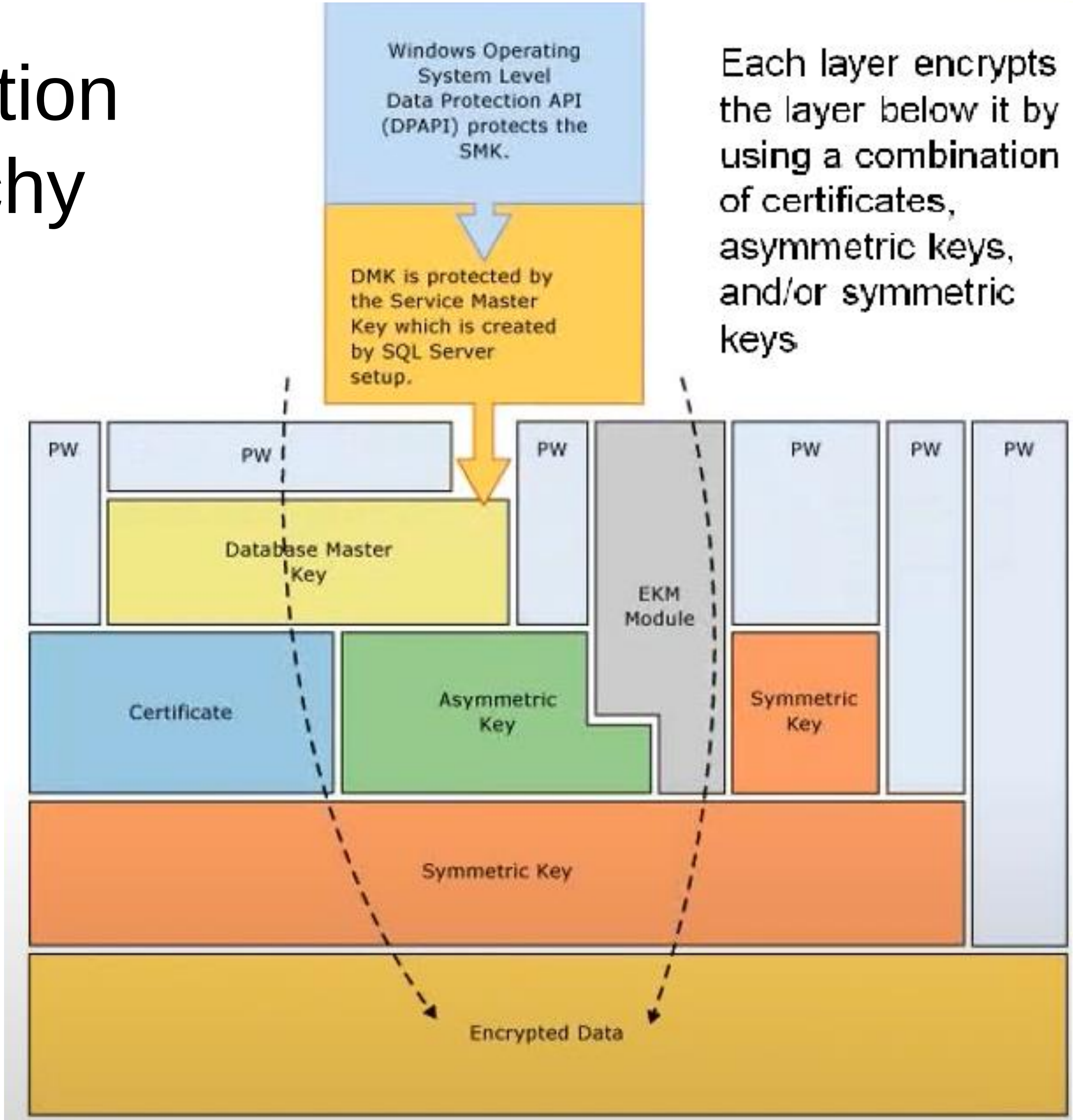
- A public key certificate, usually just called a certificate, is a digitally-signed statement that binds the value of a public key to the identity of the person, device, or service that holds the corresponding private key.
- Certificates are issued and signed by a certification authority (CA). The entity that receives a certificate from a CA is the subject of that certificate.
- Typically, certificates contain the following information:
  - The public key of the subject.
  - The identifier information of the subject, such as the name and e-mail address.
  - The validity period. This is the length of time that the certificate is considered valid.

Database Encryption

- A process that uses an algorithm to transform data in a database into **ciphertext**

- The purpose of database encryption
  - is to protect the data stored in a database from being accessed by individuals with potentially "malicious" intentions.
  - To reduce the incentive for individuals to hack the database as "meaningless" encrypted data is of little to no use for hackers.
- **Using MS – SQL we can encrypt the whole database (called TDE) and/or specific columns in table (called column level encryption)**

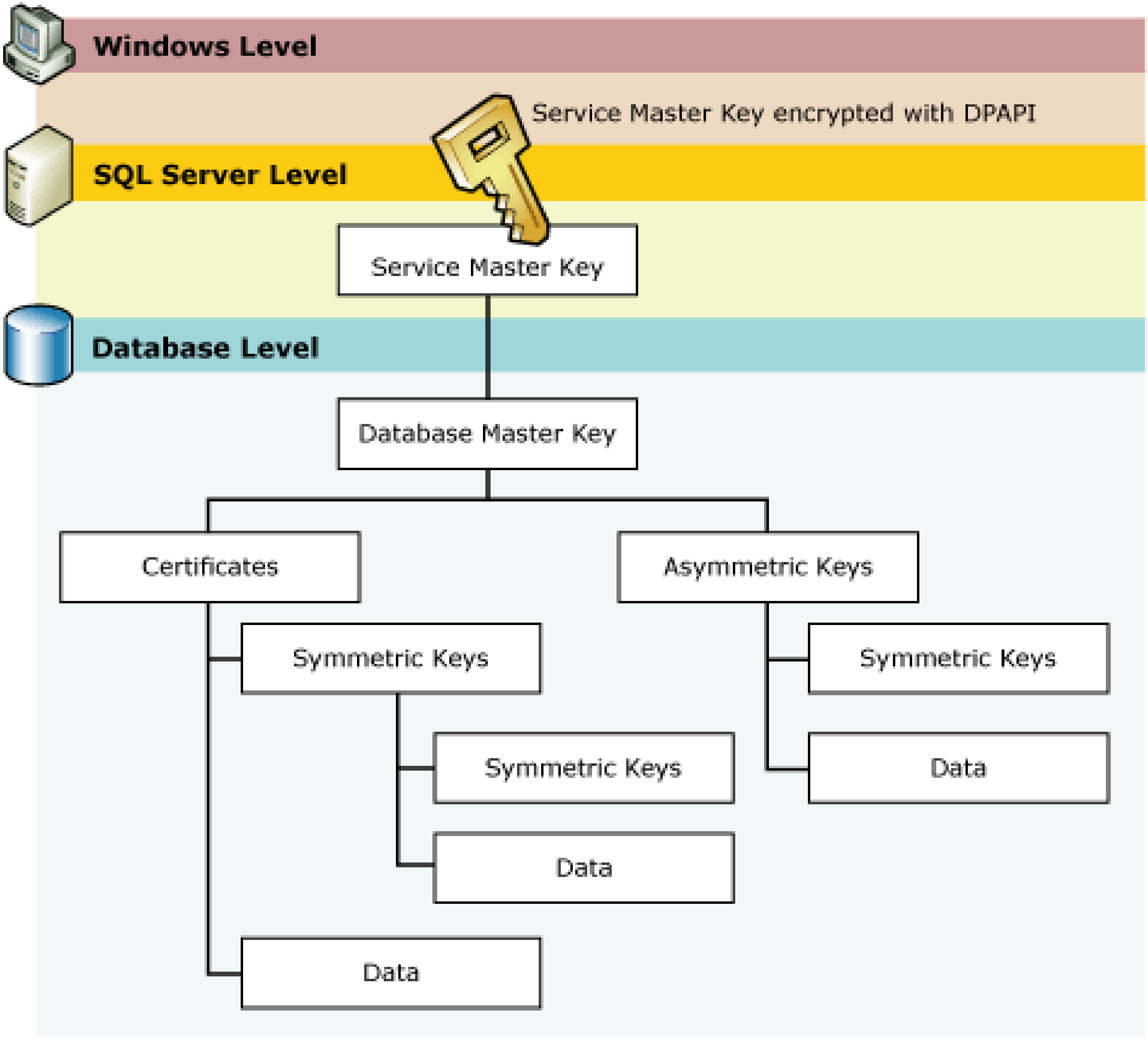
Encryption Hierarchy


Windows Operating System Level Data Protection API (DPAPI) protects the SMK.
DMK is protected by the Service Master Key which is created by SQL Server setup.
PW
PW
PW
PW
PW
PW
Database Master Key
EKM Module
Certificate
Asymmetric Key
Symmetric Key
Symmetric Key
Encrypted Data

Each layer encrypts the layer below it by using a combination of certificates, asymmetric keys, and/or symmetric keys

The Encryption Hierarchy

A · P · U
ASIA PACIFIC UNIVERSITY
OF TECHNOLOGY & INNOVATION



Service Master Key (SMK)

- SMK is the root of SQL Server's Encryption Hierarchy.
- It is a symmetric key
- Once the SQL Server is installed successfully, Service Master Key (SMK) will be automatically created.
- Service Master Key (SMK), which is the key that protects the instance. Only one SMK per SQL Server instance.
- It takes care of encrypting the user logins, passwords etc.
- Service level master key is encrypted using DPAPI.
- The service master key is used to protect (encrypt) other keys, mainly the database master keys. It cannot be used directly to encrypt data.

Database Master Key (DMK)

- DMK is the base encryption key inside of a database.
- It is a symmetric key
- There can be a DMK in each database that you have, including the master database.
- For some features, such as **TDE**, you must create a DMK in the master database.
- For others, you would create a DMK inside of the user database.
- To use the DMK, an **account needs the CONTROL permission on the database.**
- It is a symmetric key that is stored in its database and that can be used to protect certificates and asymmetric keys.
- The database master key itself is protected with one or more passwords and additionally with the service master key.

Create the Database Master Key (DMK)

```
use <db name>
go
create master key encryption by password =
'QwErTy12345!@#$%'
go
select * from sys.symmetric_keys
go
```

Creating Certificate

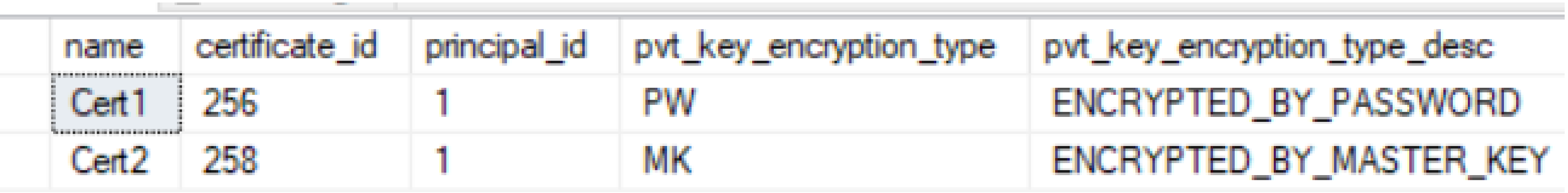
Several ways to create certificate keys

- With Password (means that it is protected by a password you specify)

```
Create Certificate Cert1
Encryption By Password = 'QwErTy12345!@#$%' With Subject = 'Cert1'
go
```

- Without password – means linking with Master Key

```
Create Certificate Cert2 With Subject = 'Cert2’
Go

select * from sys.certificates
```



name	certificate_id	principal_id	pvt_key_encryption_type	pvt_key_encryption_type_desc
Cert1	256	1	PW	ENCRYPTED_BY_PASSWORD
Cert2	258	1	MK	ENCRYPTED_BY_MASTER_KEY

Creating Asymmetric Key

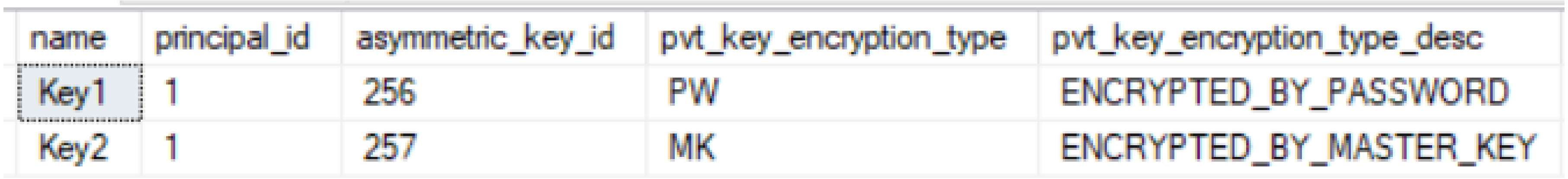
Several ways to create asymmetric keys

- With Password

```
Create Asymmetric Key Key1 With Algorithm = RSA_2048
Encryption By Password = 'QwErTy12345!@#$%'
go
```

- Without password – means linking with Master Key

```
Create Asymmetric Key Key2 With Algorithm = RSA_2048
Go

select * from sys.asymmetric_keys
```



name	principal_id	asymmetric_key_id	pvt_key_encryption_type	pvt_key_encryption_type_desc
Key1	1	256	PW	ENCRYPTED_BY_PASSWORD
Key2	1	257	MK	ENCRYPTED_BY_MASTER_KEY

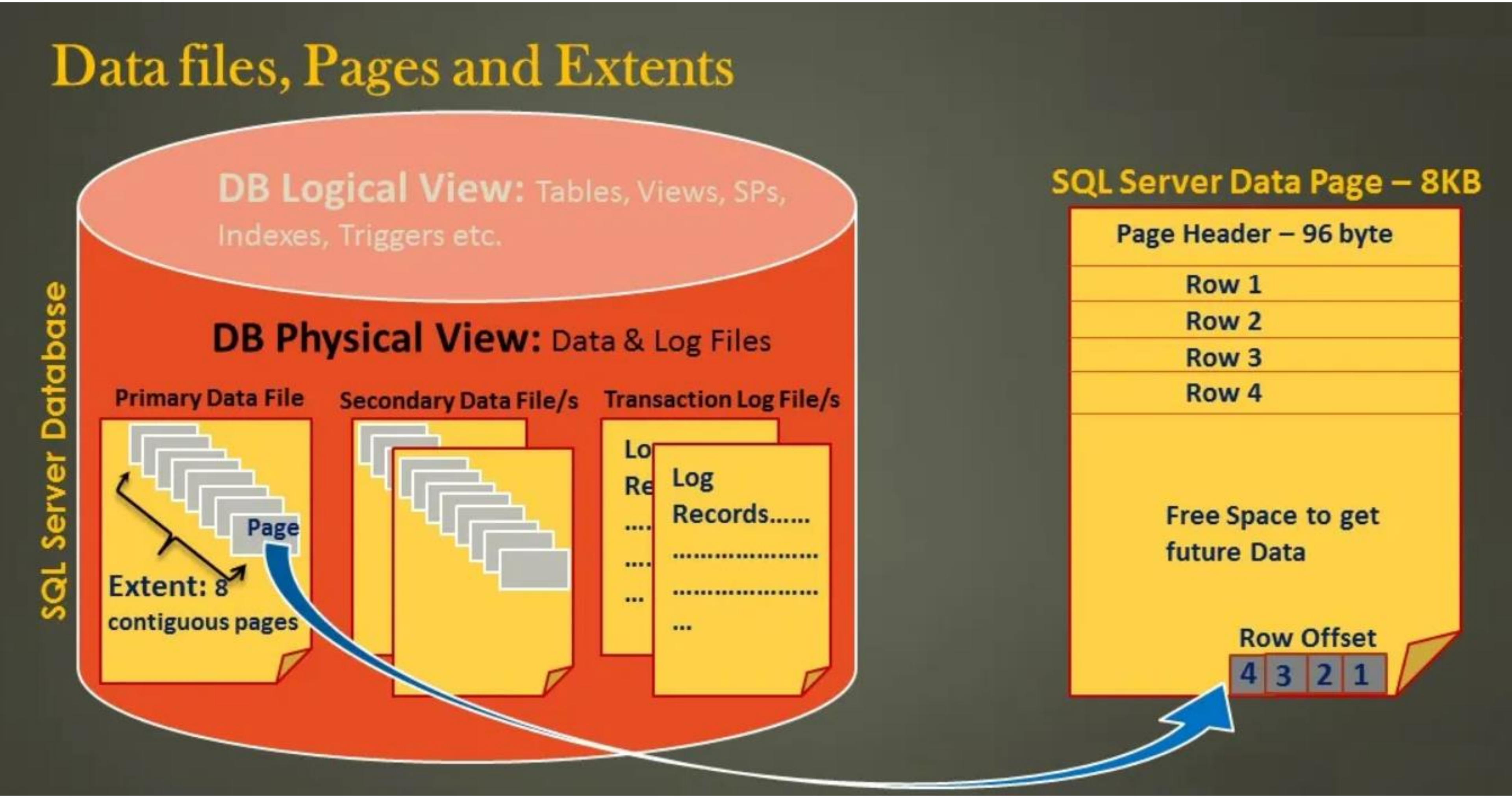
Creating Symmetric Keys

```sql
Use HospitalInfoSys
go

CREATE SYMMETRIC KEY SimKey1
WITH ALGORITHM = AES_256
ENCRYPTION BY CERTIFICATE Cert2
GO

CREATE SYMMETRIC KEY SimKey2
WITH ALGORITHM = AES_256
ENCRYPTION BY Password = 'QwErTy12345!@#$%'
GO

CREATE SYMMETRIC KEY SimKey3
WITH ALGORITHM = AES_256
ENCRYPTION BY Asymmetric Key Key2

select * from sys.symmetric_keys
```

Transparent Data Encryption (TDE)

- TDE = database level encryption
- Transparent ?? Invisible to user, No schema / design changes required
- Also known as Encryption “At Rest”
- Encryption of the database file is done at the page level. The pages in an encrypted database are encrypted before they're written to disk and are decrypted when read into memory.
- TDE doesn't increase the size of the encrypted database.
- Slight degradation of performance around 3-5%
- To enable TDE, we need to create a special symmetric key called Database Encryption Key (DEK)

Enabling Encryption - TDE


Data files, Pages and Extents
SQL Server Database
DB Logical View: Tables, Views, SPs, Indexes, Triggers etc.
DB Physical View: Data & Log Files
Primary Data File
Secondary Data File/s
Transaction Log File/s
Page
Extent: 8 contiguous pages
Lo Re ....
Log Records...... ....
SQL Server Data Page – 8KB
Page Header – 96 byte
Row 1
Row 2
Row 3
Row 4
Free Space to get future Data
Row Offset
4
3
2
1

Enabling Encryption - TDE

```
--create certificate to perform encryption
--instead of cert create keys

Use master
go
Create Certificate CertMasterDB
With Subject = 'CertmasterDB'
go

--go to the actual db that you want to encrypt
Use <db name>
go
CREATE DATABASE ENCRYPTION KEY
   WITH ALGORITHM = AES_128
   ENCRYPTION BY SERVER CERTIFICATE CertMasterDB;
go
ALTER DATABASE <db name>
SET ENCRYPTION ON;

--check database encryption status
Use master
select b.name as [DB Name], a.encryption_state_desc, a.key_algorithm,
a.encryptor_type
from sys.dm_database_encryption_keys a
inner join sys.databases b on a.database_id = b.database_id
where b.name = 'HospitalInfoSys'
```

Column Level Encryption

- Column level encryption provides a more granular control on which column to encrypt
- More secure as :-
  - each column can have its own unique encryption key within the database
  - Encryption is possible when data is active and not just “at rest”
- However, it requires changes in the code / commands
- Two types of column level encryption
  - **Database side**
  - Client side – out of scope our scope
    - Utilize “Always Encrypted” setting

Encrypt Column Data

- Let’s say we need to capture patient’s payment card numbers and it must be encrypted.
- This columns must be defined as **varbinary** datatype

```
PaymentCardNo varbinary(MAX)
```

Create Encryption Key

```sql
CREATE SYMMETRIC KEY SimKey1
WITH ALGORITHM = AES_256
ENCRYPTION BY CERTIFICATE Cert2
GO
```

Adding Encrypted Data

If we are using symmetric key, we need to open the key during encryption and decryption. Then insert the data using EncryptByKey function

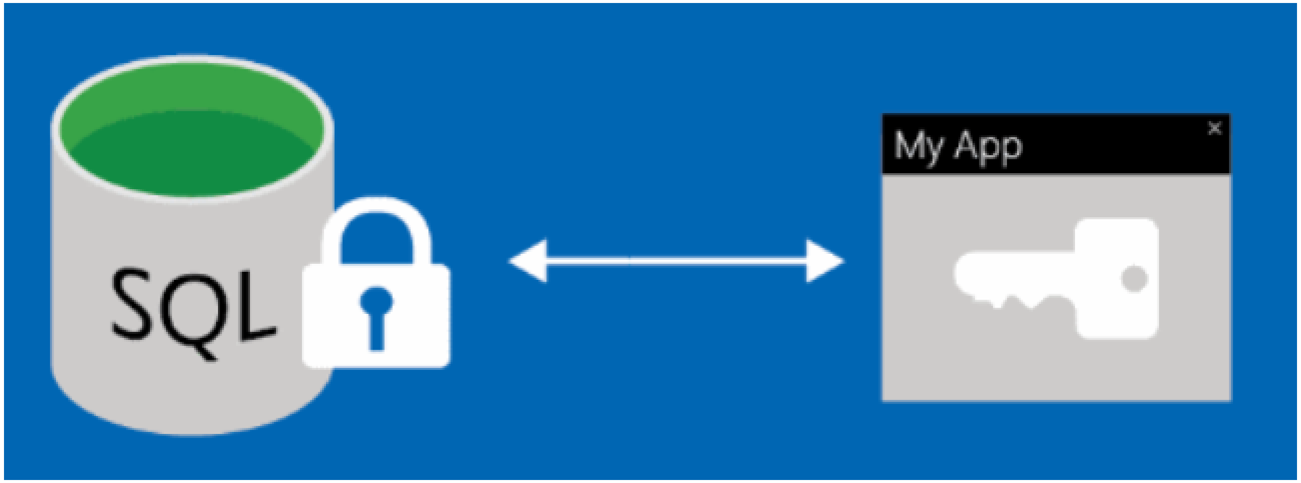
```
OPEN SYMMETRIC KEY SimKey1
DECRYPTION BY CERTIFICATE Cert2

INSERT INTO [dbo].[Patient]
([PatientNo],[PatientName],[Address],[Telephone],[DOB],
 [Gender],[RegistrationDate],[MaritalStatus],
 [PaymentCardNo])
VALUES ('Pat300', 'Sam', 'Penang','0134567899',getdate()-8500,
'Male',getdate()-250, 'Single',
EncryptByKey(Key_GUID('SimKey1'),'123456789’))

CLOSE SYMMETRIC KEY SimKey1
```

Reading Encrypted Data

- To get back the encrypted data (decrypt it), we need to use the DecryptByKey function

```
OPEN SYMMETRIC KEY SimKey1
DECRYPTION BY CERTIFICATE Cert2

Select  [PatientNo],[PatientName],[Address],[Telephone],[DOB],
[Gender],[RegistrationDate],[MaritalStatus],
CONVERT(varchar, DecryptByKey(PaymentCardNo)),
from Patient
```

Always Encrypted

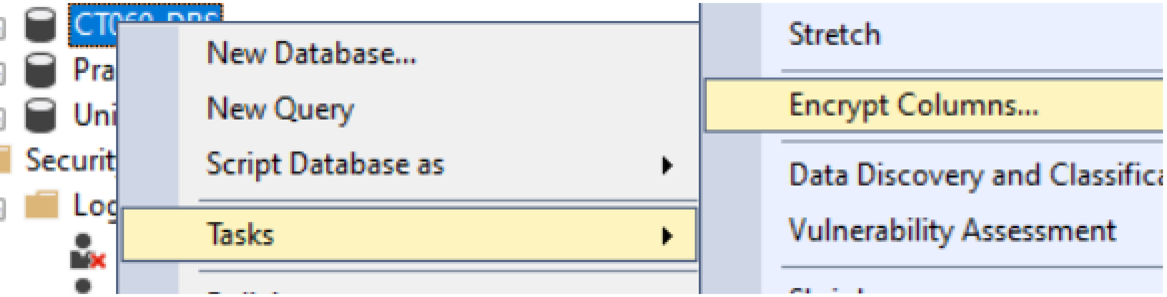


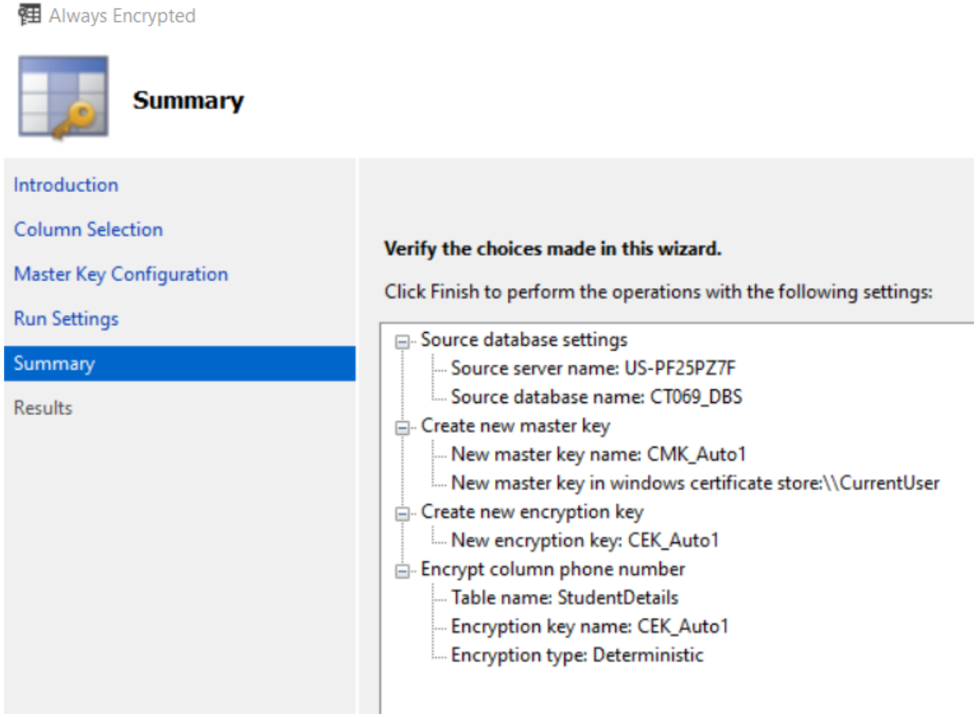
Always Encrypted is a feature designed to protect sensitive data, such as credit card numbers or national identification numbers stored in SQL Server databases.

Always Encrypted

- Always Encrypted allows clients to encrypt sensitive data inside client applications and never reveal the encryption keys to the SQL Database Engine
- As a result, Always Encrypted provides a separation between those who own the data and can view it, and those who manage the data but should have no access.
- By ensuring on-premises database administrators or other high-privileged unauthorized users, can't access the encrypted data, Always Encrypted enables customers to confidently store sensitive data outside of their direct control.
- Further reading: https://docs.microsoft.com/en-us/sql/relational-databases/security/encryption/always-encrypted-client-development?view=sql-server-ver15

Always Encrypted





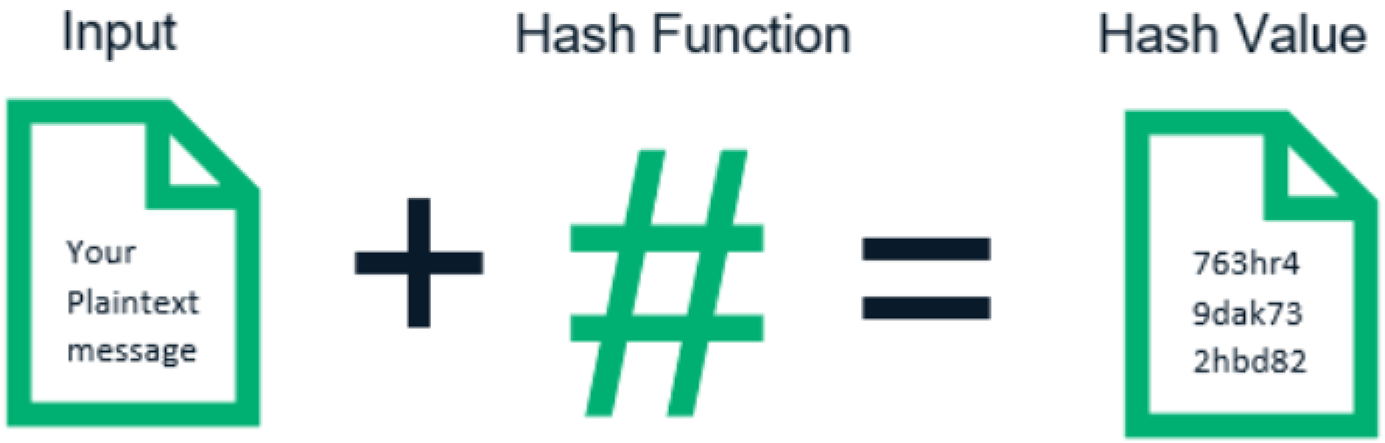
Verify the choices made in this wizard.

Click Finish to perform the operations with the following settings:

- Source database settings
  - Source server name: US-PF25PZ7F
  - Source database name: CT069_DBS
- Create new master key
  - New master key name: CMK_Auto1
  - New master key in windows certificate store:\\CurrentUser
- Create new encryption key
  - New encryption key: CEK_Auto1
- Encrypt column phone number
  - Table name: StudentDetails
  - Encryption key name: CEK_Auto1
  - Encryption type: Deterministic

Hashing

- Hashing is the process of converting an input of variable length to a fixed size array of numbers and letters using a mathematical function.



Hashing .. continued

- Each hash value or output must be unique.
- It is immutable in the sense that the same input will produce the exact same hash.
- A hash function needs to be secure. Even a slight change to the input file should produce a vastly different hash value.
- It is irreversible, i.e., it’s not possible to arrive at the original input file from its hash value.

Encryption	Hashing
Encryption is a two-way function where information is scrambled using an encryption key and unscrambled later using a decryption key.	Hashing is a one-way function where a unique message digest is generated from an input file or a string of text. No keys are used.
The message is encoded in a way that only authorized parties can access it. It’s used to prevent unauthorized users from reading data from a file by rendering it into an unreadable form.	Hashing is the process of using hash functions on data to map it to a fixed size output. It’s like a checksum and is used for verifying file integrity. Hashing is useful where you want to compare an entered value with a stored value without needing to read the original content of the file.
The resultant encrypted string is of a variable length.	The resultant hashed string is of a fixed length.
The original message can always be retrieved by using the appropriate decryption key.	Output can’t be reverted to the original message. The best hashing algorithms are designed in a way that makes it virtually impossible to retrieve the original string from the hash value.
There are two primary types of encryption: Symmetric key encryption (or private key encryption) and Asymmetric key encryption (or public key encryption) Examples of encryption algorithms: RSA, AES, DES, etc.	Examples of hashing algorithms: SHA-1, SHA-2, MD5, Tiger, etc.
Purpose of encryption is to transmit data securely (i.e., protect data confidentiality)	The objective of using hashing is to verify data (i.e., protect data integrity)

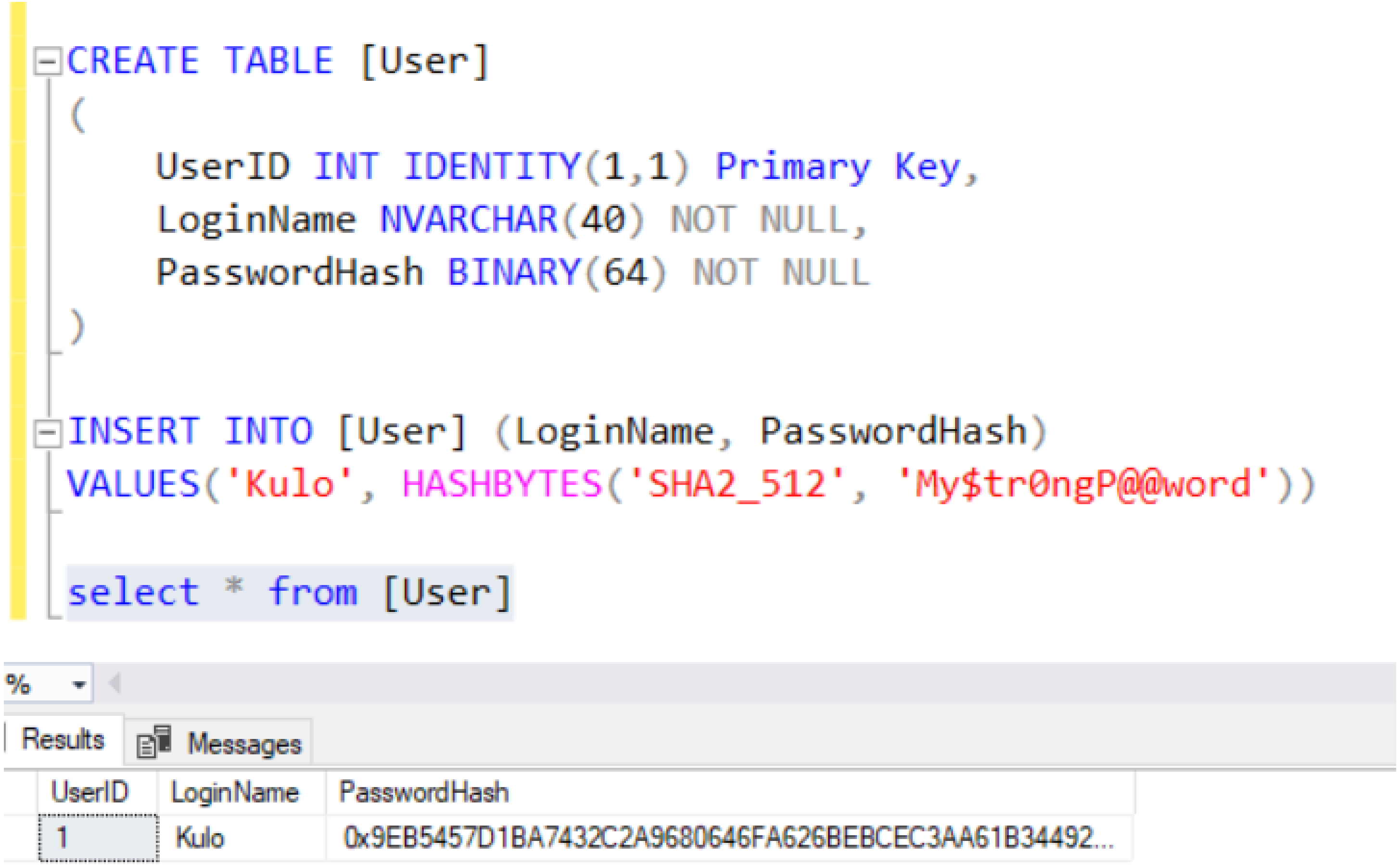
Security Use Case for Hashing in SQL

- Storing passwords
  - Hashing is almost always preferable to encryption when storing passwords inside databases because in the event of a compromise attackers won't get access to the plaintext passwords
  - The has value a non-reversible cryptographic representation of the password
- Recreating Logins
  - We can recreate logins if needed without knowing the actual password by retrieving the hash value from SQL

Use Case for Hashing in SQL

- Identifying incremental data or changed data
  - Hash values generated for an entire row (by concatenating values of all the columns of the row and then generating hash key on it) are useful for efficiently searching for differences between rows in tables and identifying if the rows have changed, in case there is no mechanism in place to identify incremental data on the source table.
- Faster Search
  - Hashing is used to index and retrieve items in a database
  - It is faster to find the item using the shorter hashed key than to find it using the original value.

Storing Password Securely



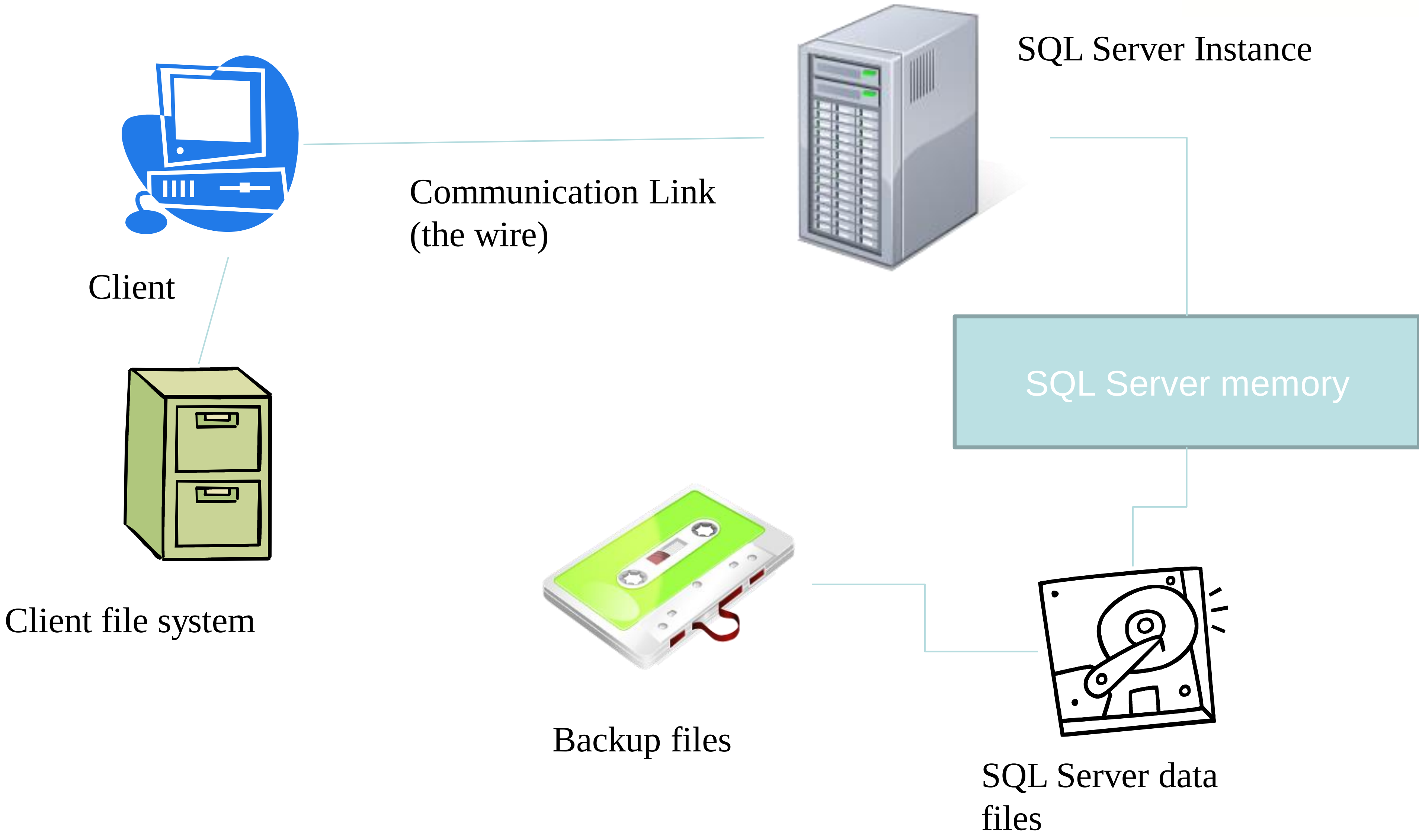
```sql
CREATE TABLE [User]
(
    UserID INT IDENTITY(1,1) Primary Key,
    LoginName NVARCHAR(40) NOT NULL,
    PasswordHash BINARY(64) NOT NULL
)

INSERT INTO [User] (LoginName, PasswordHash)
VALUES('Kulo', HASHBYTES('SHA2_512', 'My$tr0ngP@@word'))

select * from [User]
```

	UserID	LoginName	PasswordHash
	1	Kulo	0x9EB5457D1BA7432C2A9680646FA626BEBCEC3AA61B34492...

Recreating Logins

```
SELECT password_hash FROM sys.sql_logins
WHERE name = <login id>

CREATE LOGIN <login id> WITH PASSWORD = <hash
value> HASHED
```

Encryption in SQL Server



SQL Server Instance

Communication Link
(the wire)

Client

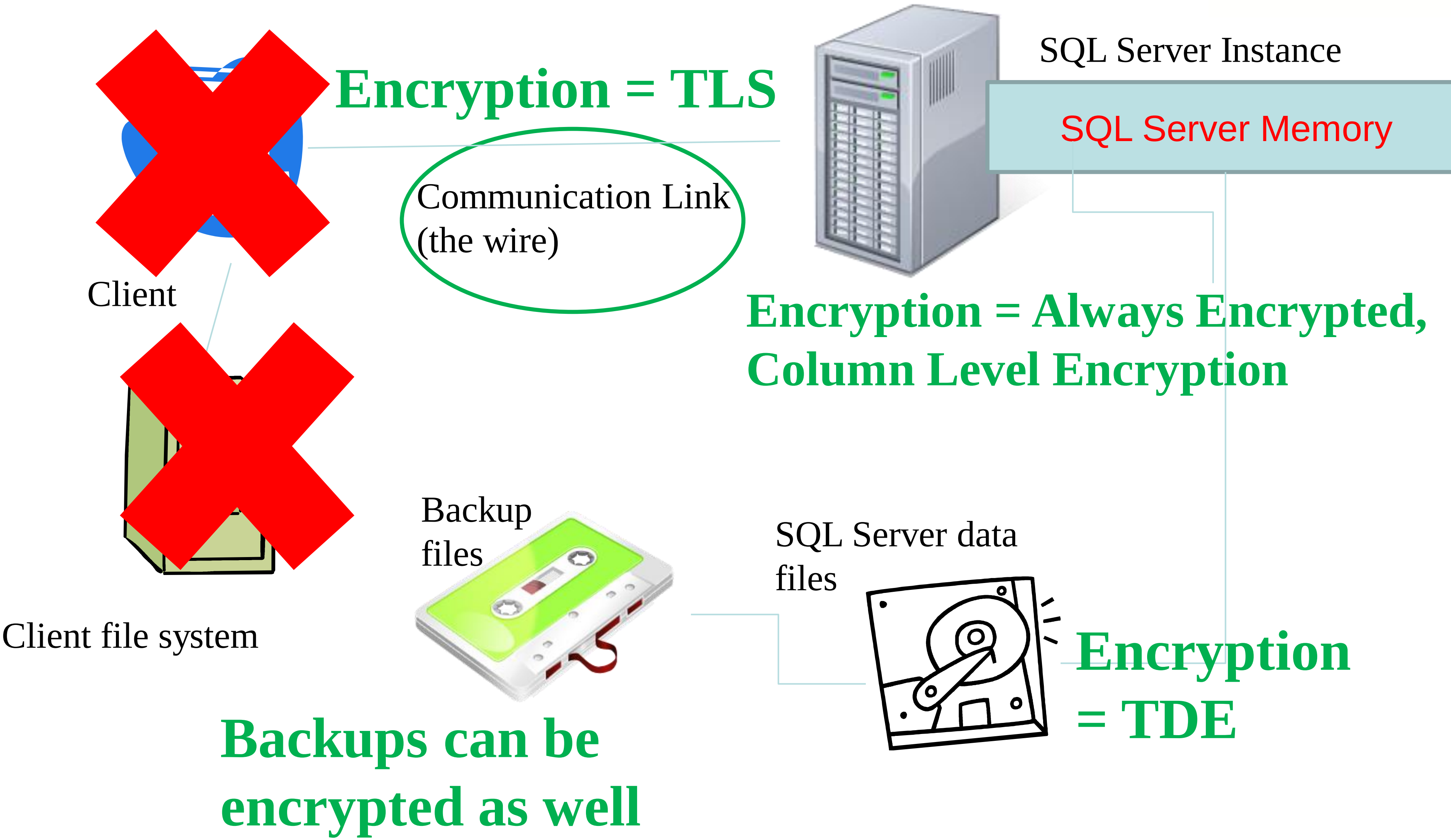
SQL Server memory

Client file system

Backup files

SQL Server data
files

Encryption in SQL Server



SQL Server Instance

SQL Server Memory

Encryption = TLS

Communication Link (the wire)

Client

Client file system

Encryption = Always Encrypted, Column Level Encryption

Backup files

SQL Server data files

Encryption = TDE

Backups can be encrypted as well

Q & A