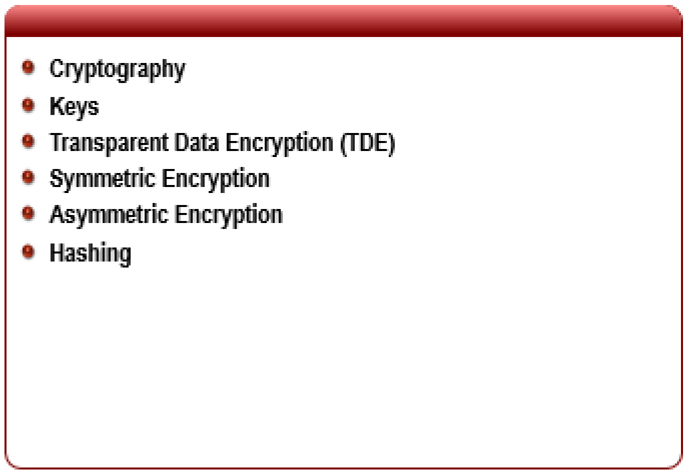
Module 4:
Protecting Data

Contents

Module Overview



- Cryptography
- Keys
- Transparent Data Encryption (TDE)
- Symmetric Encryption
- Asymmetric Encryption
- Hashing

This module is one of the most important modules in this course. This is because when all security elements fail (i.e. installation errors, authentication, authorization, bad access policy), all that is left is the data. In other words, if database records containing critical information are not protected, then all security elements are irrelevant.

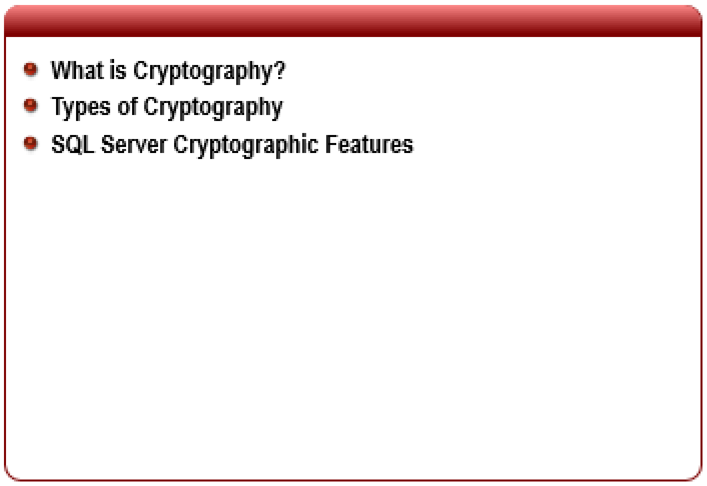
This lesson is about how to protect data even if a system is compromised and/or overtaken.

Objectives

After completing this module, you will be able to:

- Understand what is cryptography
- What are the cryptographic elements
- Define cryptography in SQL Server
- Recognize and use correct cryptographic SQL Server mechanisms

Lesson 1: Cryptography



- What is Cryptography?
- Types of Cryptography
- SQL Server Cryptographic Features

We use cryptography every day: on the internet, mobile devices, ATM machines, and in almost every aspects of our digital life.

In a nutshell, cryptography is about data scrambling and hiding, depending on the implementation and user-specific needs.

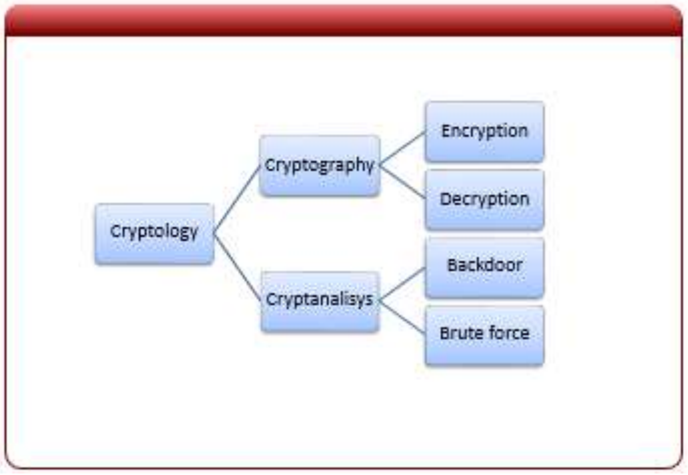
A database is the spine of every information system and is the specific target of potential data thieves. SQL Server has one of the best cryptographic set of features that we can use to create a state of the art security and privacy “aware” systems.

Objectives

After completing this lesson, you will be able to:

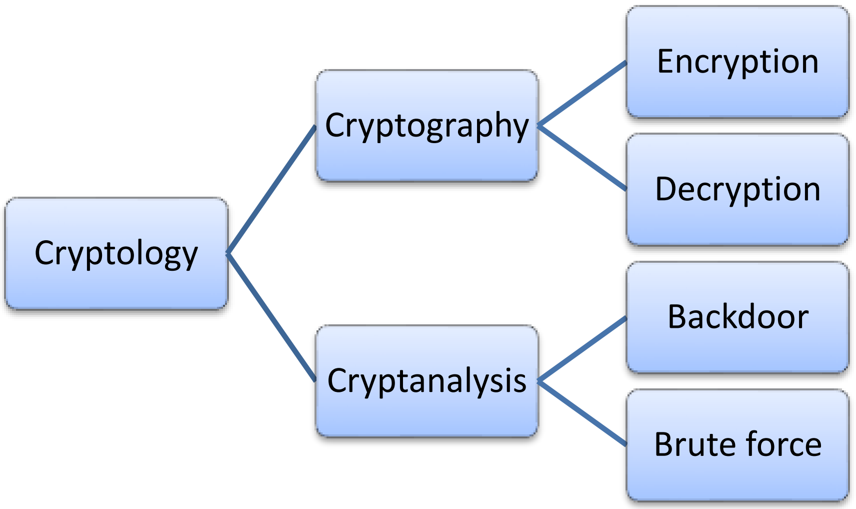
- Understand cryptography
- Distinguish different types of cryptography
- Identify SQL Server cryptographic features

What is Cryptography?



Cryptology is an art and science based in mathematic theory with the purpose of creating secret codes. It has two major components:

- Cryptography
- Cryptanalysis



Cryptography is about creating a secret codes, while cryptanalysis is the process of breaking secret codes.

Cryptography has two terms:

- Encryption – process of creating an obfuscated message from a plain text using a key.
- Decryption – process of returning plain text from an obfuscated message using a key.

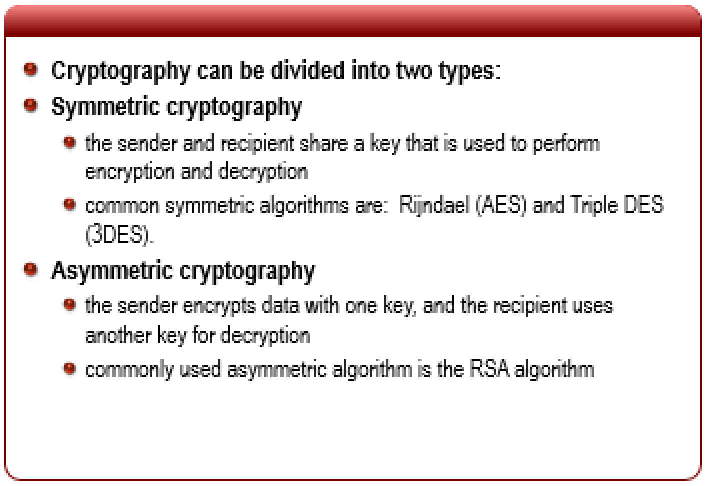
Cryptanalysis has two components:

-
- Backdoor – a cryptosystem has a backdoor for accessing plain text from encrypted messages without the regular process of decryption.
- Brute force – the only way to decrypt a message, aside from the regular decryption process, is by testing all possible combinations.

This lesson is about using built-in techniques for encryption database cells, records and whole databases.

Cryptography is not a new way of hiding data; it has roots in ancient history. The first known encryption technique was used in ancient Rome, and is known as “Caesar cipher”. This encryption technique involved basic letter substitution. Utilizing plain text alphabet, the encrypted version of a message involved the shifting of each letter by three (3). For example, letter A is D, B is E, C is F, etc. The process of encryption was simple; write normal text then exchange the original letters with the shifted correspondents.

Types of Cryptography



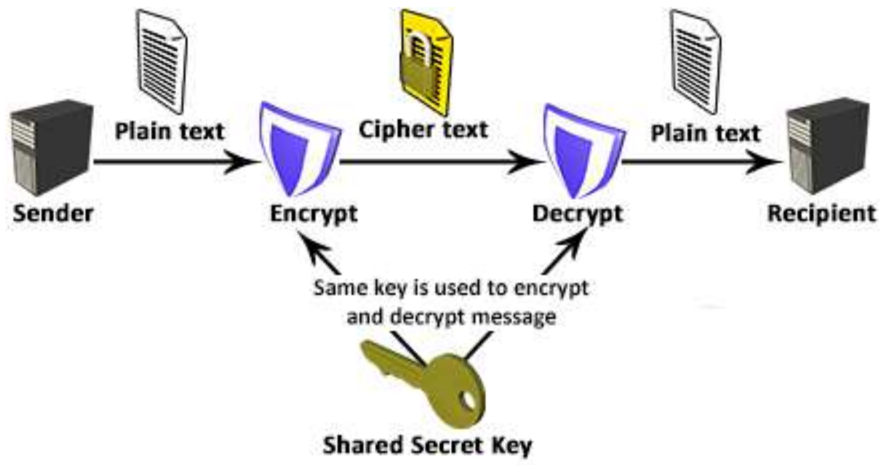
- Cryptography can be divided into two types:
- Symmetric cryptography
  - the sender and recipient share a key that is used to perform encryption and decryption
  - common symmetric algorithms are: Rijndael (AES) and Triple DES (3DES).
- Asymmetric cryptography
  - the sender encrypts data with one key, and the recipient uses another key for decryption
  - commonly used asymmetric algorithm is the RSA algorithm

Looking at the development of cryptography since ancient Rome (Project Venona, the Zimmerman telegraph, Enigma, Data Encryption Standard, and the RSA algorithm), it can be divided into two types:

- Symmetric cryptography
- Asymmetric cryptography

Symmetric Cryptography

In symmetric cryptography cases, the sender and recipient share a key that is used to perform encryption and decryption. Symmetric cryptography is the most popular way for encryption in modern IT.

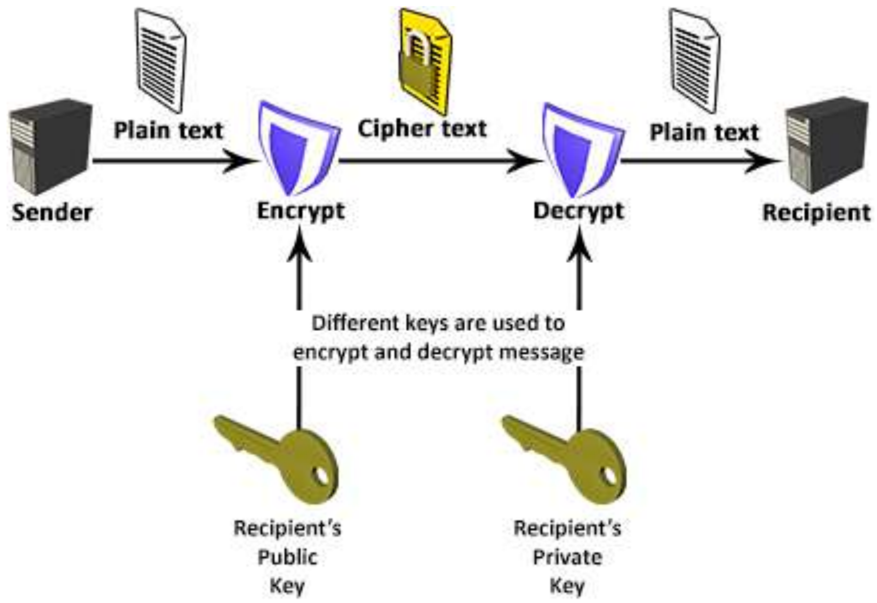


Some of the most common symmetric algorithms are: Rijndael (AES) and Triple DES (3DES).

Symmetric cryptography is simple because the same key that is used for encryption and decryption. But, before communication can occur, the sender and the recipient must exchange a secret key. The exchange of the shared secret key is the only weakness of symmetric cryptography.

Asymmetric Cryptography

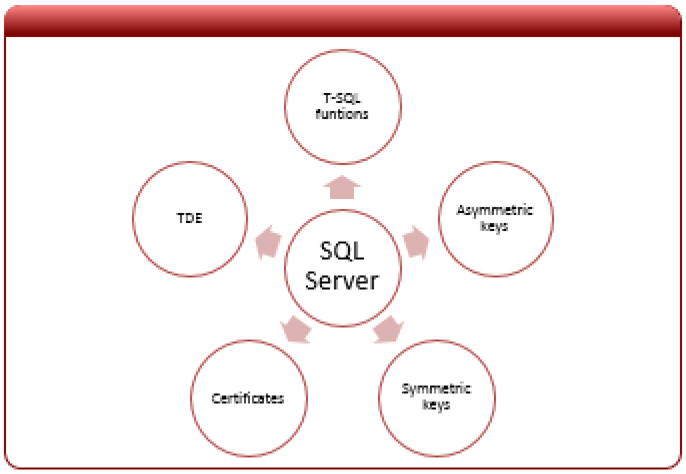
With asymmetric cryptography (also known as public key cryptography), the sender encrypts data with one key, and the recipient uses another key for decryption. The encryption and decryption key are known to us as a public/private key pair.



The most commonly used asymmetric algorithm is the RSA algorithm.

Asymmetric encryption requires more processing power than symmetric encryption. Because of this, asymmetric encryption is usually optimized by adding a symmetric key to encrypt a message and then asymmetrically encrypting the shared key. This can reduce the amount of data that is asymmetrically encrypted and also improves performance.

SQL Server Cryptographic Features



As we defined before, encryption is the process of obfuscating data by the use of a key or password. This can make the data useless without the corresponding decryption key or password. Encryption does not solve access control problems. However, it enhances security by limiting data loss even if access controls are bypassed. For example, if the database host computer is misconfigured and a hacker obtains sensitive data, that stolen information might be useless if it is encrypted.

SQL Server provides the following mechanisms for encryption:

- Transact-SQL functions
- Asymmetric keys
- Symmetric keys
- Certificates
- Transparent Data Encryption

Transact-SQL Functions

Individual items can be encrypted as they are inserted or updated using Transact-SQL functions.

Certificates

A public key certificate is a digitally-signed statement that connects data of a public key to the identity of the person, device, or service that holds the private key. Certificates are issued and signed by a Certification Authority (CA).

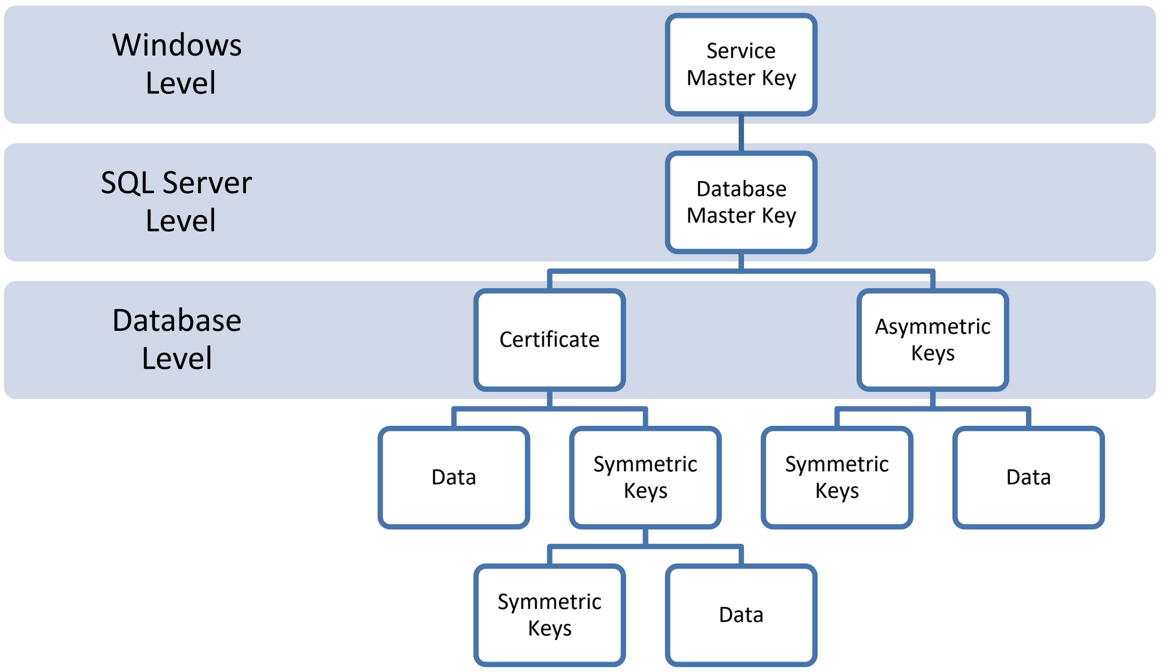
Transparent Data Encryption

Transparent Data Encryption (TDE) is a special type of encryption which uses a symmetric key called the database encryption key.

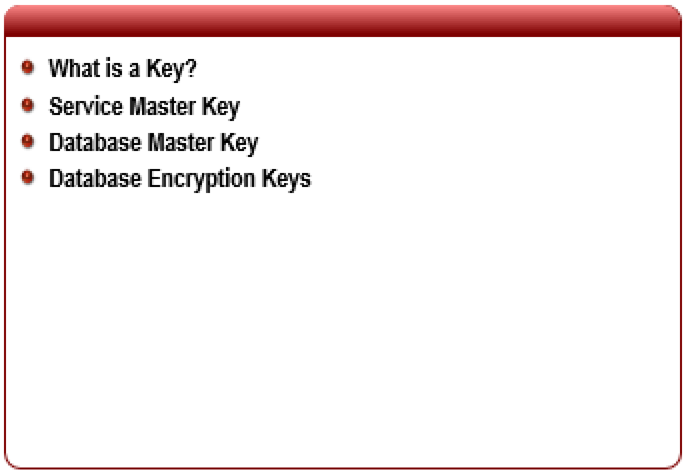
Symmetric and asymmetric keys, which were discussed earlier in this module, can be used to encrypt data in a database.

Encryption Hierarchy

SQL Server encrypts data with a hierarchical encryption. Each layer encrypts the layer below it using certificates, asymmetric keys, and symmetric keys. Asymmetric and symmetric keys can be stored outside of SQL Server in an Extensible Key Management (EKM) module.



Lesson 2: Keys



- What is a Key?
- Service Master Key
- Database Master Key
- Database Encryption Keys

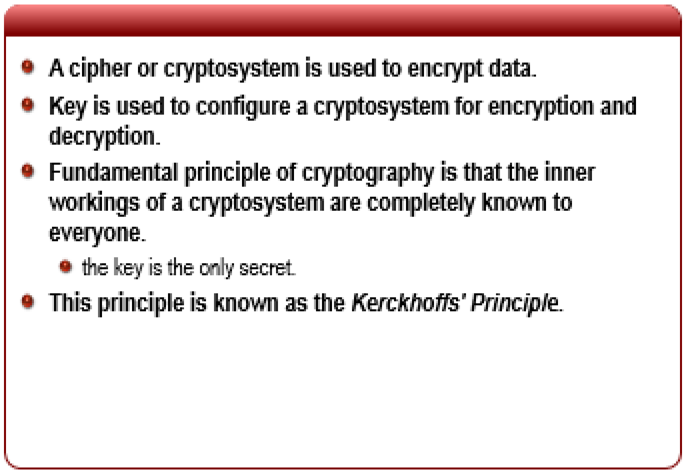
Proper encryption requires appropriate understanding, creation and use of keys. Keys are the major and most sensitive part of the encryption process. This lesson will teach you basic concepts of encryption keys and how to use such keys in SQL Server.

Objectives

After completing this lesson, you will be able to:

- Define a key
- Understand a Service Master Key
- Create a Database Master Key
- Understand database encryption keys

What is a Key?



- A cipher or cryptosystem is used to encrypt data.
- Key is used to configure a cryptosystem for encryption and decryption.
- Fundamental principle of cryptography is that the inner workings of a cryptosystem are completely known to everyone.
  - the key is the only secret.
- This principle is known as the *Kerckhoffs' Principle*.

A cipher or cryptosystem is used to encrypt data. A key is used to configure a cryptosystem for encryption and decryption. A fundamental principle of cryptography is that the inner workings of a cryptosystem are completely known to everyone. However, the key is the only secret. This principle is known as the *Kerckhoffs' Principle*.

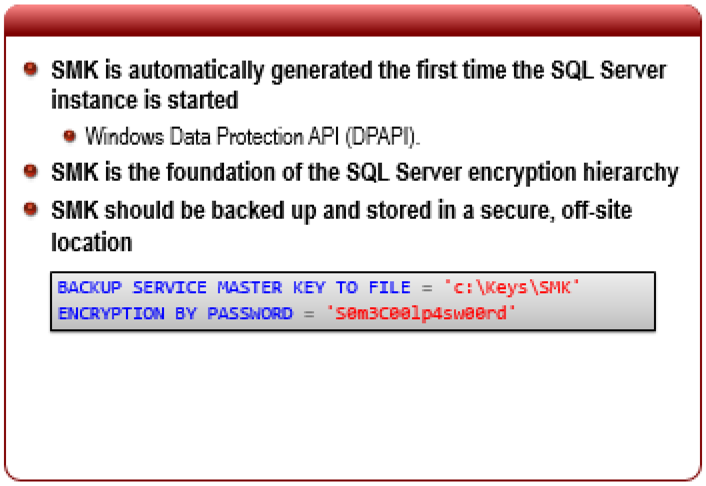
This same principle is found in SQL Server. SQL Server implements well known cryptographic solutions called algorithms. Types of algorithms, such as specification, source codes, etc., are readily available online for free.

A key is the product of a specific cryptosystem and is based on randomly collected information, such as random numbers, the temperature of the CPU, sample data in RAM, etc. The randomly collected information is entered into a cryptosystem which then generates a key. A key is hard to handle by users because it is long and contains hard readable data. Due to its complexity, a cryptosystem will associate a key with a password. In most cases, the password will trigger the key to start the encryption/decryption process.

In cryptography, the key size or length is measured in bits. A longer key means a more secure system. However, a longer key will affect performance because the encryption process takes longer. Therefore, it is important to choose an appropriate type of encryption and key length.

Both symmetric and asymmetric keys are measured in bits. Despite this similarity, symmetric and asymmetric keys are different. For example, a symmetric key using AES can be 256 bits long, while an asymmetric key using RSA can be as long as 2048 bits. Although 2048 bits may appear more secure than 256 bits, it does not mean that RSA is more secure than AES. Both RSA and AES are different and not comparable. For example, the security available with a 1024-bit key using asymmetric RSA is considered approximately equal in security to an 80-bit key using a symmetric algorithm.

Service Master Key



- SMK is automatically generated the first time the SQL Server instance is started
  - Windows Data Protection API (DPAPI).
- SMK is the foundation of the SQL Server encryption hierarchy
- SMK should be backed up and stored in a secure, off-site location

```
BACKUP SERVICE MASTER KEY TO FILE = 'c:\Keys\SMK'
ENCRYPTION BY PASSWORD = 'S0m3C00lp4sw00rd'
```

SQL Server has two primary applications for keys: a Service Master Key (SMK) generated on and for a SQL Server instance, and a Database Master Key (DMK) used for a database.

The SMK is automatically generated the first time the SQL Server instance is started and is used to encrypt a linked server password, credentials, and the DMK. The SMK is encrypted by using the local computer key which uses the Windows Data Protection API (DPAPI).

The DPAPI uses a key that is derived from the Windows credentials of the SQL Server service account and the computer's credentials. The SMK can only be decrypted by the service account under which it was created or by a principal that has access to the machine's credentials.

If you need to change service account, it is recommended to use the SQL Server Configuration Manager. To manage a change of the service account, SQL Server stores a redundant copy of the SMK protected by the machine account.

```
ALTER SERVICE MASTER KEY REGENERATE;
GO
```

This T-SQL code regenerates a SMK. When the SMK is regenerated, the SQL Server decrypts all the keys that have been encrypted with it, and then encrypts them with the new SMK. Keep in mind that the regeneration of a SMK is a resource-intensive operation. Consequently, you should schedule this operation during “off” hours, unless the system has been compromised. If any of the decryptions fail, the whole statement fails.

The SMK should be backed up and stored in a secure, off-site location.

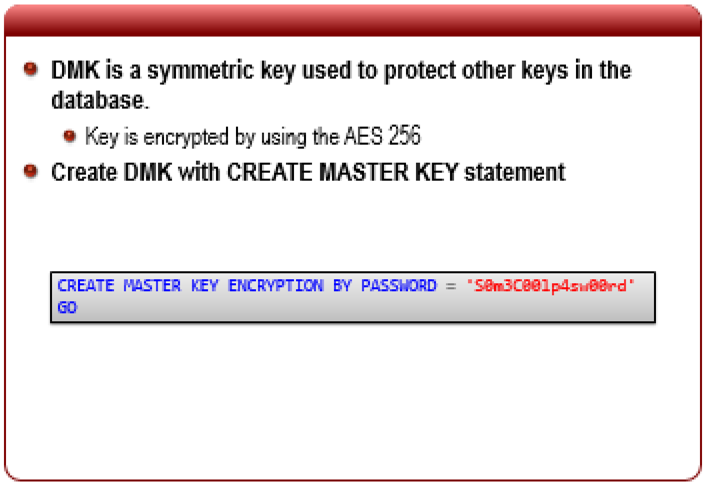
```
BACKUP SERVICE MASTER KEY TO FILE = 'c:\Keys\SMK'
ENCRYPTION BY PASSWORD = 'S0m3C00lp4sw00rd'
```

When the SMK is restored, the SQL Server decrypts all the keys and data that have been encrypted with the current SMK, and then encrypts them with the SMK from the backup.

```
RESTORE SERVICE MASTER KEY
   FROM FILE = 'c:\Keys\SMK'
   DECRYPTION BY PASSWORD = 'S0m3C00lp4sw00rd';
GO
```

The SMK is the foundation of the SQL Server encryption hierarchy. The SMK directly or indirectly protects all other keys and data in the encryption hierarchy tree. If a dependent key cannot be decrypted during a forced regeneration, the data that the key secures will be lost.

Database Master Key



- DMK is a symmetric key used to protect other keys in the database.
  - Key is encrypted by using the AES 256
- Create DMK with CREATE MASTER KEY statement

```
CREATE MASTER KEY ENCRYPTION BY PASSWORD = 'S0m3C001p4sw00rd'
GO
```

The Database Master Key (DMK) is a symmetric key used to protect the private keys of certificates and asymmetric keys that are present in the database. When it is created, the master key is encrypted by using the AES_256 algorithm and a user-supplied password. To enable the automatic decryption of the master key, a copy of the key is encrypted by using the SMK and stored in both the database (user and in the master database). Copy stored in the master is always updated whenever the master key is changed.

This default can be changed by using the DROP ENCRYPTION BY SERVICE MASTER KEY option of ALTER MASTER KEY. A master key that is not encrypted by the SMK must be opened by using the OPEN MASTER KEY statement and a password.

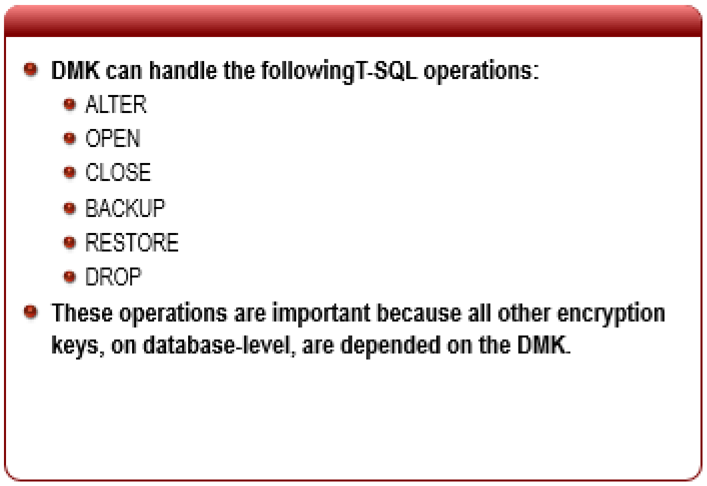
We can create a database master with the CREATE MASTER KEY statement.

```
CREATE MASTER KEY ENCRYPTION BY PASSWORD = 'S0m3C001p4sw00rd'
GO
```

SQL Server can generate, for each user database, a different DMK. This is a good feature because it makes the database more secure.

You should backup the master key by using BACKUP MASTER KEY and store the backup in a secure and off-site location.

Database Encryption Keys



- DMK can handle the followingT-SQL operations:
  - ALTER
  - OPEN
  - CLOSE
  - BACKUP
  - RESTORE
  - DROP
- These operations are important because all other encryption keys, on database-level, are depended on the DMK.

Now that we know how the DMK is important and how to create one, we will continue with the following DMK operations:

- ALTER
- OPEN
- CLOSE
- BACKUP
- RESTORE
- DROP

These operations are important because all other encryption keys, on database-level, are depended on the DMK.

We can easily create a new DMK for AdventureWorks2012 and re-encrypt the keys below it in the encryption hierarchy, assuming that we have the DMK created in the lesson before.

```
USE AdventureWorks2012
GO
ALTER MASTER KEY REGENERATE
WITH ENCRYPTION BY PASSWORD = 'S0m3C00lp4sw00rdforNewK3y'
GO
```

If the DMK was encrypted with the SMK, it will be automatically opened when it is needed for decryption or encryption. In this case, it is not necessary to use the OPEN MASTER KEY statement.

Opening the DMK for use:

```
USE AdventureWorks2012
GO
OPEN MASTER KEY
DECRYPTION BY PASSWORD = 'S0m3C00lp4sw00rdforNewK3y'
GO
```

Closing the DMK after use:

```
USE AdventureWorks2012
GO
CLOSE MASTER KEY
GO
```

The master key must be open before it is backed up.

Backing up the DMK:

```
USE AdventureWorks2012
GO
OPEN MASTER KEY
DECRYPTION BY PASSWORD = 'S0m3C00lp4sw00rdforNewK3y';
BACKUP MASTER KEY TO FILE = 'c:\Keys\DMK'
ENCRYPTION BY PASSWORD = '4jfmdn48ndno20';
GO
```

When the master key is restored, SQL Server decrypts all the keys that are encrypted with the currently active master key, and then encrypts these keys with the restored master key.

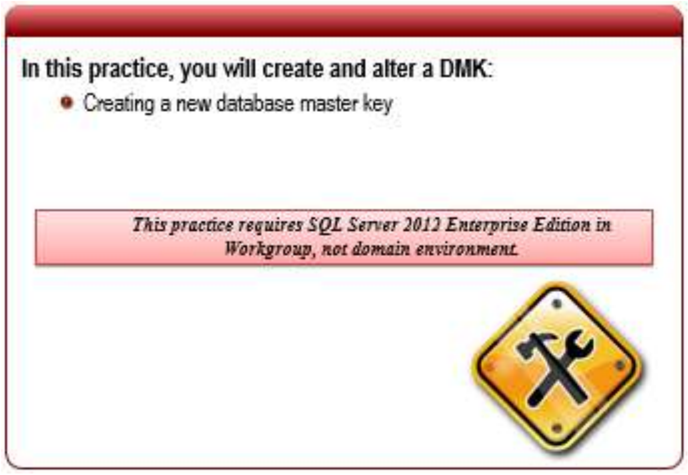
Restoring the DMK:

```
USE AdventureWorks2012
GO
RESTORE MASTER KEY
FROM FILE = 'c:\Keys\DMK'
DECRYPTION BY PASSWORD = '4jfmdn48ndno20'
ENCRYPTION BY PASSWORD = 'S0m3C00lp4sw00rdforNewK3y';
GO
```

Dropping the DMK:

```
USE AdventureWorks2012
GO
DROP MASTER KEY
GO
```

Practice: Creating a Database Master Key


In this practice, you will create and alter a DMK:

- Creating a new database master key

This practice requires SQL Server 2012 Enterprise Edition in Workgroup, not domain environment.

In this practice, you will create and alter a DMK.

> ***This practice requires SQL Server 2012 Enterprise Edition in Workgroup, not domain environment.***

Exercise 1: Creating a new database master key

In this exercise, you will create a new database and a DMK as a starting cryptographic element.

1. In **SSMS** open a new query window.

2. Create a new database:

```
CREATE DATABASE DMKTest
GO
```

3. Create new DMK:

```
USE DMKTest
GO
CREATE MASTER KEY
ENCRYPTION BY PASSWORD = 'P4$$w0rd'
GO
```

4. Query the system catalog to check the DMK:

```
SELECT * FROM sys.symmetric_keys
```

5. Alter the DMK so that is not encrypted by the SMK anymore:

```
USE DMKTest
```

```
GO
ALTER MASTER KEY
DROP ENCRYPTION BY SERVICE MASTER KEY
GO
```

6. Restore the environment to its original state:

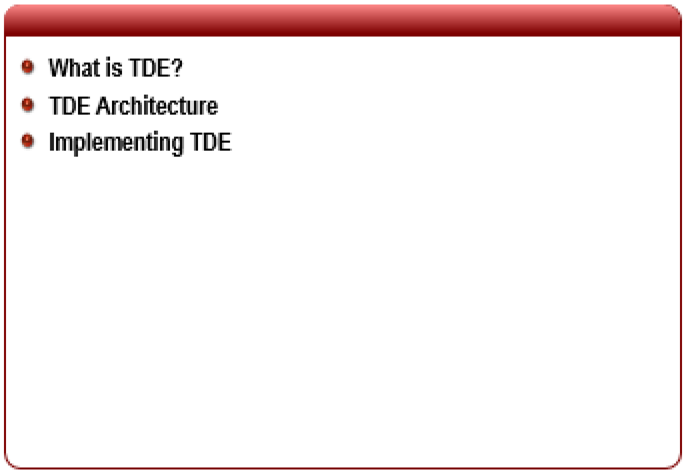
```
USE DMKTest
GO
DROP MASTER KEY
GO

DROP DATABASE DMKTest
GO
```

7. Query the system catalog to check if the DMK is dropped (the query should return 0 (zero) rows).

```
SELECT * FROM sys.symmetric_keys
```

Lesson 3: Transparent Data Encryption (TDE)



- What is TDE?
- TDE Architecture
- Implementing TDE

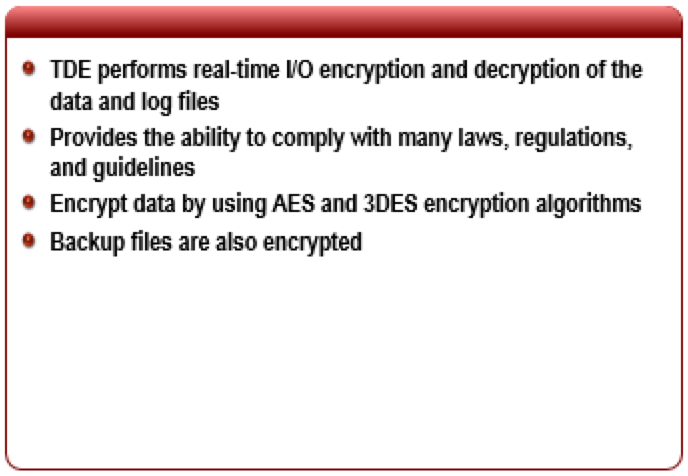
SQL Server has two ways of encrypting data. One way is by protecting data on the table, record or column level, and the other way is by protecting data “at the rest”. One of the best crypto features in the database world today is known as a Transparent Data Encryption.

Objectives

After completing this lesson, you will be able to:

- Define TDE
- Understand TDE architecture
- Implement TDE

What is TDE?



- TDE performs real-time I/O encryption and decryption of the data and log files
- Provides the ability to comply with many laws, regulations, and guidelines
- Encrypt data by using AES and 3DES encryption algorithms
- Backup files are also encrypted

Imagine the following scenario. Someone has unauthorized access to your database system environment. That person finds a way to get the last database backup file, copies it and takes it in an unsecured environment. In this moment, the security mechanism just fell apart.

This scenario illustrates what can happen when someone illegally copies, detaches, and restores your database. The consequences for such activity can be substantial, depending on the sensitivity of your data environment.

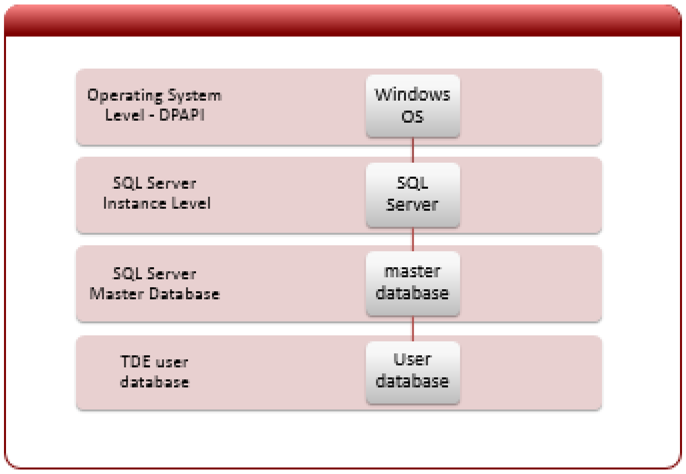
Those unpleasant situations can be avoided by using Transparent Database Encryption (TDE).

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 or an asymmetric key protected by an EKM module. It provides the ability to comply with many laws, regulations, and guidelines established in various industries. This enables software developers to encrypt data by using AES and 3DES encryption algorithms without changing existing applications. TDE does not provide encryption across communication channels.

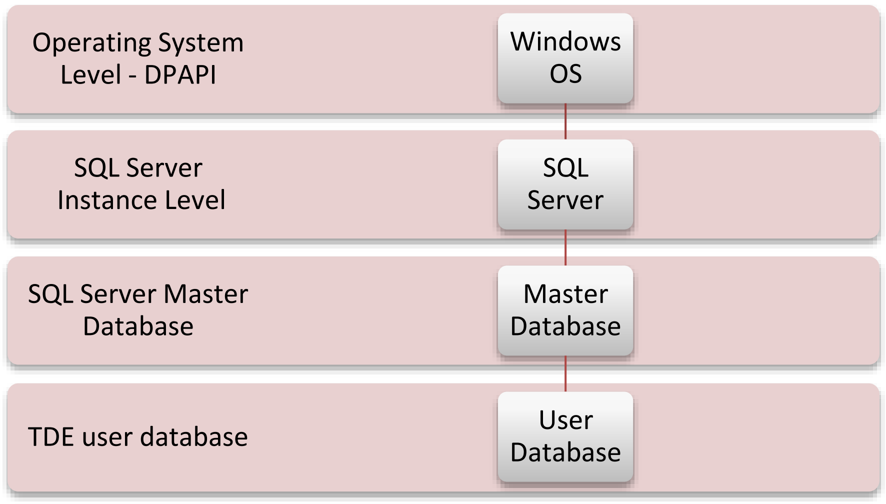
Backup files of databases that have TDE enabled are also encrypted by using the DEK. As a result, when you restore these backup files, the certificate protecting the DEK must be available. This means that in addition to backing up the database, you have to make sure that you maintain backups of the server certificates to prevent data loss. Data loss will result if the certificate is no longer available.

When enabling TDE, you should immediately backup the certificate and the private key associated with that certificate. If the certificate ever becomes unavailable, or if you must restore or attach the database on another server, you must have backups of both the certificate and the private key in order to be able to open the database.

TDE Architecture

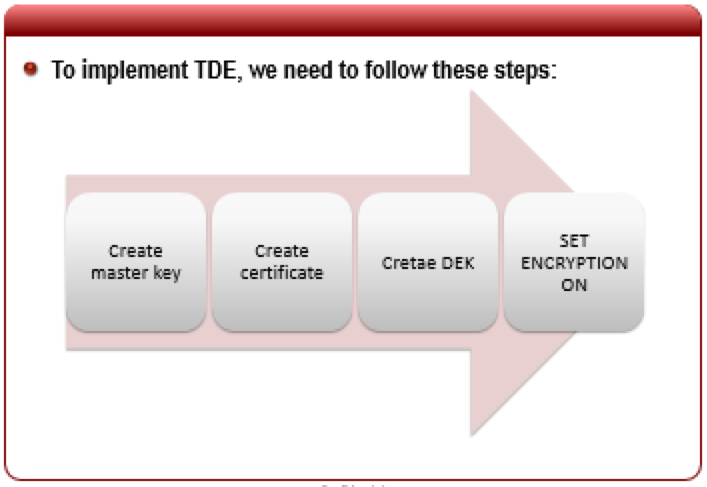

Operating System Level - DPAPI
Windows OS
SQL Server Instance Level
SQL Server
SQL Server Master Database
master database
TDE user database
User database

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. TDE does not increase the size of the encrypted database.


Operating System Level - DPAPI
Windows OS
SQL Server Instance Level
SQL Server
SQL Server Master Database
Master Database
TDE user database
User Database

DPAPI encrypts the SMK (created at the time of a SQL Server setup). SMK encrypts the DMK of the master database. DMK of the master database creates a certificate in the master database. The certificate encrypts the DMK in the user database. The entire user database is secured by DEK of the user database by using TDE.

Implementing TDE


- To implement TDE, we need to follow these steps:

Create master key
Create certificate
Cretae DEK
SET ENCRYPTION ON

To implement TDE, we need to follow these steps:

- Create a master key
- Create or obtain a certificate protected by using the master key
- Create a database encryption key and protect it by using the certificate
- Set the database to use encryption

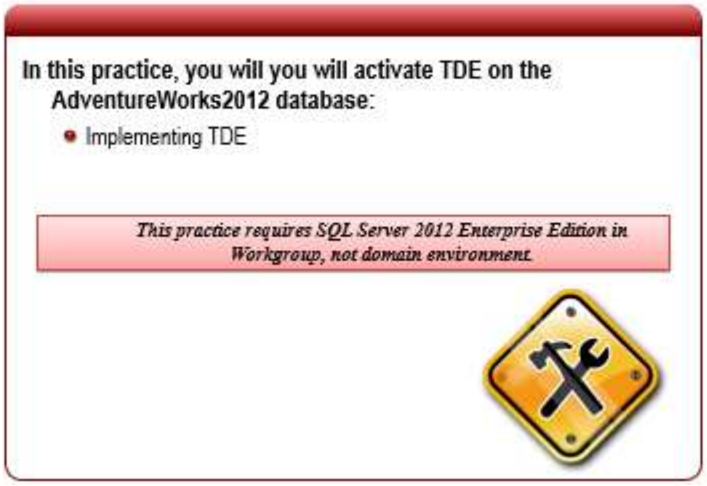
```
USE master;
GO
CREATE MASTER KEY ENCRYPTION BY PASSWORD = 'S0m3C00lp4sw00rd';
GO

CREATE CERTIFICATE MyServerCert
WITH SUBJECT = 'MyCertificate';
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
```

Practice: Implementing TDE



In this practice, you will you will activate TDE on the AdventureWorks2012 database:

- Implementing TDE

This practice requires SQL Server 2012 Enterprise Edition in Workgroup, not domain environment.

In this practice, you will activate TDE on the AdventureWorks2012 database.

This practice requires SQL Server 2012 Enterprise Edition in Workgroup, not domain environment.

Exercise 1: Implementing TDE

In this exercise, you will pass all necessary steps for activating TDE on the AdventureWorks2012 sample database.

1. Create a Master Encryption Key.

```
USE master;
GO
CREATE MASTER KEY
ENCRYPTION BY PASSWORD = 'Some3xtr4Passw00rd';
GO
```

2. Create a certificate and check it.

```
USE master
GO
CREATE CERTIFICATE TDE WITH SUBJECT = 'My TDE Certificate';
GO

SELECT * FROM sys.certificates
```

3. Create a demo database **TDEdb** and Database Encryption Key. The key is encrypted with the certificate from step 2.

```
CREATE DATABASE TDEdb
GO
```

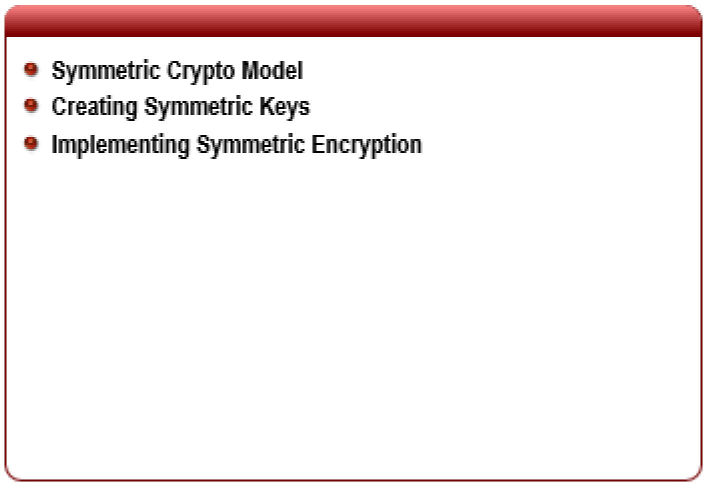
```
USE TDEdb
GO
CREATE DATABASE ENCRYPTION KEY
WITH ALGORITHM = AES_256
ENCRYPTION BY SERVER CERTIFICATE TDE;
GO
```

4. Turn on **TDE** feature.

```
ALTER DATABASE TDEdb
SET ENCRYPTION ON;
GO
```

5. Now, detach **TDEdb** database and try to attach on a different instance of SQL Server. Explain the error message.

Lesson 4: Symmetric Encryption



- Symmetric Crypto Model
- Creating Symmetric Keys
- Implementing Symmetric Encryption

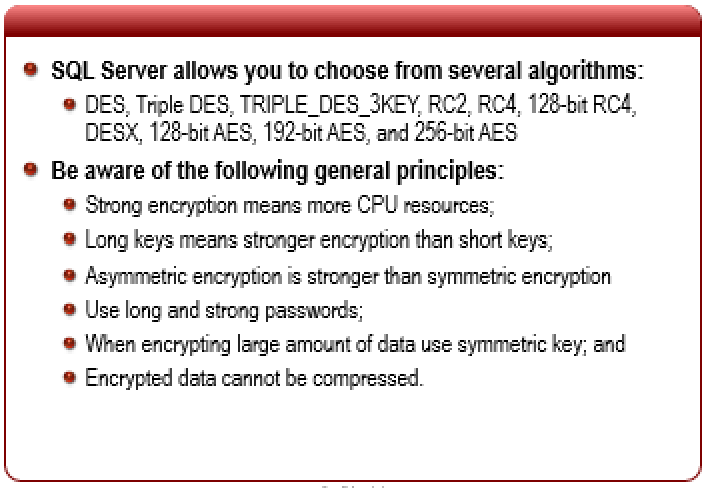
This lesson will teach you how to implement powerful symmetric cryptography elements built-in to the SQL Server core engine and T-SQL functions. Symmetric cryptography is one of the best features for protecting data in database tables.

Objectives

After completing this lesson, you will be able to:

- Understand the symmetric encryption model
- Create symmetric keys
- Implement symmetric encryption on SQL Server

Symmetric Encryption Model



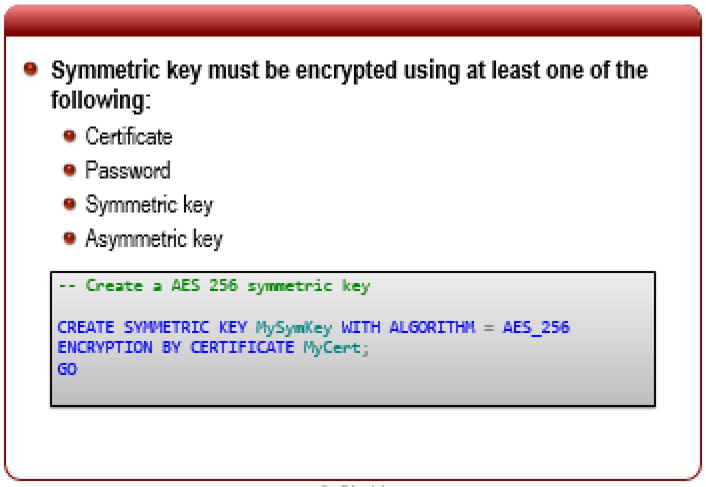
- **SQL Server allows you to choose from several algorithms:**
  - DES, Triple DES, TRIPLE_DES_3KEY, RC2, RC4, 128-bit RC4, DESX, 128-bit AES, 192-bit AES, and 256-bit AES
- **Be aware of the following general principles:**
  - Strong encryption means more CPU resources;
  - Long keys means stronger encryption than short keys;
  - Asymmetric encryption is stronger than symmetric encryption
  - Use long and strong passwords;
  - When encrypting large amount of data use symmetric key; and
  - Encrypted data cannot be compressed.

Symmetric encryption is the type of encryption that uses the same key for encryption and decryption. SQL Server allows you to choose from several algorithms, including DES, Triple DES, TRIPLE_DES_3KEY, RC2, RC4, 128-bit RC4, DESX, 128-bit AES, 192-bit AES, and 256-bit AES.

No single algorithm is ideal for all situations; however, the following general principles apply:

- Strong encryption requires more CPU resources;
- Long keys generally yield stronger encryption than short keys;
- Asymmetric encryption is stronger than symmetric encryption if using the same key size; however, performance is compromised;
- Long and strong passwords are better than short and/or weak passwords;
- If you are encrypting large amounts of data, you should encrypt using a symmetric key because of performance issues; and
- Encrypted data cannot be compressed, but compressed data can be encrypted.

Creating Symmetric Keys



- Symmetric key must be encrypted using at least one of the following:
  - Certificate
  - Password
  - Symmetric key
  - Asymmetric key

```
-- Create a AES 256 symmetric key

CREATE SYMMETRIC KEY MySymKey WITH ALGORITHM = AES_256
ENCRYPTION BY CERTIFICATE MyCert;
GO
```

When a symmetric key is created, it must be encrypted by using at least one of the following:

- Certificate
- Password
- Symmetric key
- Asymmetric key

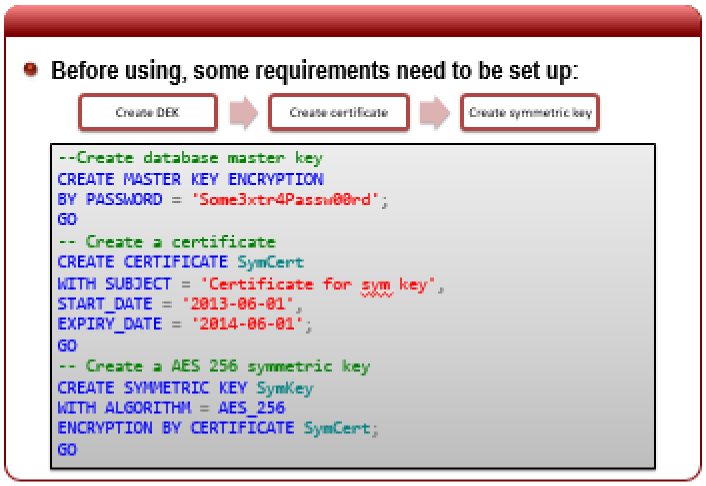
The key can have more than one encryption of each type. In other words, a single symmetric key can be encrypted by using multiple certificates, passwords, symmetric keys, and asymmetric keys at the same time.

When a symmetric key is encrypted with a password instead of the public key of the DMK, the TRIPLE DES encryption algorithm is used. Because of this, keys that are created with a strong encryption algorithm, such as AES, are themselves secured by a weaker algorithm.

```
-- Create a AES 256 symmetric key

CREATE SYMMETRIC KEY MySymKey WITH ALGORITHM = AES_256
      ENCRYPTION BY CERTIFICATE MyCert;
GO
```

Implementing Symmetric Encryption

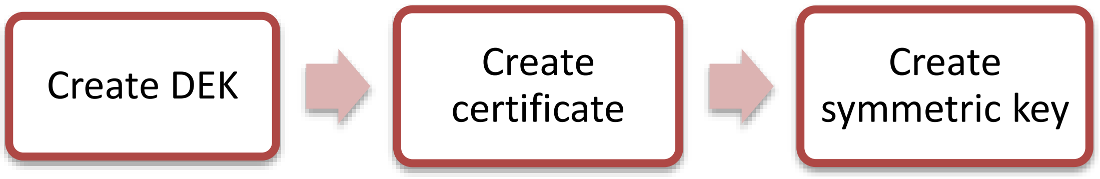


- Before using, some requirements need to be set up:

Create DEK → Create certificate → Create symmetric key

```
--Create database master key
CREATE MASTER KEY ENCRYPTION
BY PASSWORD = 'Some3xtr4Passw00rd';
GO
-- Create a certificate
CREATE CERTIFICATE SymCert
WITH SUBJECT = 'Certificate for sym key',
START_DATE = '2013-06-01',
EXPIRY_DATE = '2014-06-01';
GO
-- Create a AES 256 symmetric key
CREATE SYMMETRIC KEY SymKey
WITH ALGORITHM = AES_256
ENCRYPTION BY CERTIFICATE SymCert;
GO
```

Before we can implement symmetric encryption, there are some requirements that need to be set up:

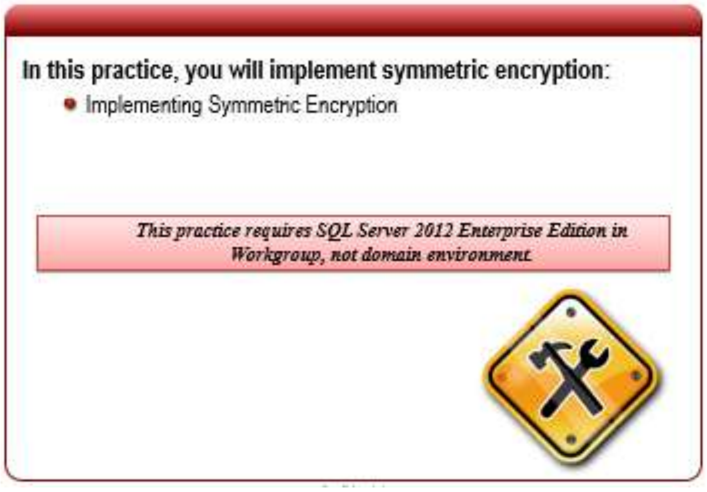


```
--Create database master key
CREATE MASTER KEY ENCRYPTION
BY PASSWORD = 'Some3xtr4Passw00rd';
GO

-- Create a certificate
CREATE CERTIFICATE SymCert
WITH SUBJECT = 'Certificate for sym key',
START_DATE = '2013-06-01',
EXPIRY_DATE = '2014-06-01';
GO

-- Create a AES 256 symmetric key
CREATE SYMMETRIC KEY SymKey
WITH ALGORITHM = AES_256
ENCRYPTION BY CERTIFICATE SymCert;
GO
```

Practice: Implementing Symmetric Encryption


In this practice, you will implement symmetric encryption:

- Implementing Symmetric Encryption

This practice requires SQL Server 2012 Enterprise Edition in Workgroup, not domain environment.

In this practice, you will implement symmetric encryption.

This practice requires SQL Server 2012 Enterprise Edition in Workgroup, not domain environment.

Exercise 1: Implementing Symmetric Encryption

In this exercise, you will pass all necessary steps for implementing symmetric encryption in a user-created sample database. Also, you will encrypt data in a user table.

1. Create sample database.

```
CREATE DATABASE SymCryptDB
GO
USE SymCryptDB
GO
```

2. Create DMK.

```
CREATE MASTER KEY
ENCRYPTION BY PASSWORD = 'Some3xtr4Passw00rd';
GO
```

3. Create certificate and symmetric key.

```
CREATE CERTIFICATE SymCert
WITH SUBJECT = 'Certificate for sym key',
START_DATE = '2013-06-01',
EXPIRY_DATE = '2014-06-01';
GO
```

```
CREATE SYMMETRIC KEY SymKey
WITH ALGORITHM = AES_256
ENCRYPTION BY CERTIFICATE SymCert;
GO
```

4. Create sample table:

```
CREATE TABLE EncryptedCustomer
(
  CustomerID   int NOT NULL PRIMARY KEY,
  FirstName    varbinary(200),
  MiddleName   varbinary(200),
  LastName     varbinary(200),
  EmailAddress varbinary(200),
  Phone        varbinary(150),
  rowguid      uniqueidentifier
);
```

5. Open the key that is protected by the certificate.

```
OPEN SYMMETRIC KEY SymKey
DECRYPTION BY CERTIFICATE SymCert;
GO
```

6. Insert the sample from AdventureWorksLT2012 and encrypt the data.

```
INSERT INTO EncryptedCustomer
(
  CustomerID,
  FirstName,
  MiddleName,
  LastName,
  EmailAddress,
  Phone,
  rowguid
)
SELECT
  CustomerID,
  EncryptByKey(Key_Guid(N'SymKey'), FirstName),
  EncryptByKey(Key_Guid(N'SymKey'), MiddleName),
  EncryptByKey(Key_Guid(N'SymKey'), LastName),
  EncryptByKey(Key_Guid(N'SymKey'), EmailAddress),
  EncryptByKey(Key_Guid(N'SymKey'), Phone),
  rowguid
FROM AdventureWorksLT2012.SalesLT.Customer;
GO
```

7. Close the key.

```
CLOSE SYMMETRIC KEY SymKey;
GO
```

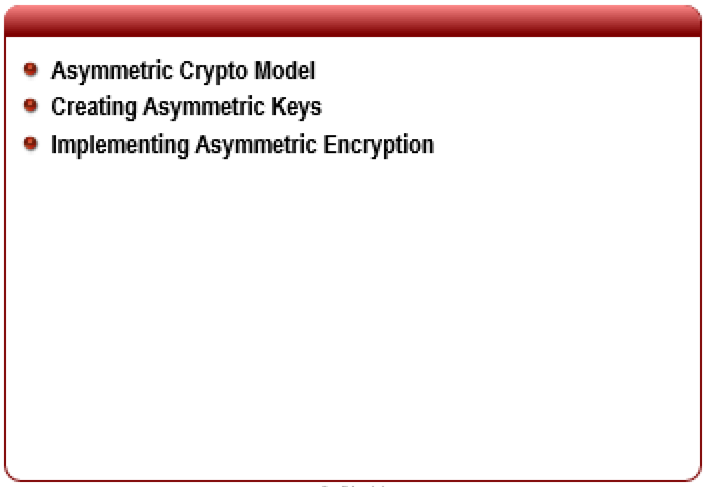
8. Check the encrypted data.

```
SELECT
  *
FROM EncryptedCustomer;
GO
```

9. Decrypt the sample data.

```
OPEN SYMMETRIC KEY SymKey
DECRYPTION BY CERTIFICATE SymCert;
GO
SELECT
 CustomerID,
 CAST(DecryptByKey(FirstName) AS nvarchar(100)) AS
 DecryptedFirstName,
 FirstName
FROM EncryptedCustomer;
GO
```

Lesson 5: Asymmetric Encryption


- Asymmetric Crypto Model
- Creating Asymmetric Keys
- Implementing Asymmetric Encryption

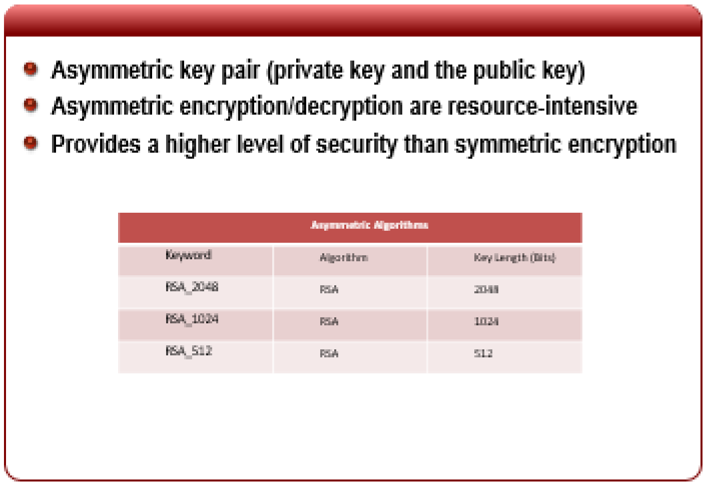
This lesson will teach you how to implement asymmetric cryptography elements built-in to the SQL Server core engine and T-SQL functions. Asymmetric cryptography is as secure as symmetric cryptography, but the key length is significantly larger.

Objectives

After completing this lesson, you will be able to:

- Understand the asymmetric encryption model
- Create asymmetric keys
- Implement asymmetric encryption on SQL Server

Asymmetric Encryption Model



- Asymmetric key pair (private key and the public key)
- Asymmetric encryption/decryption are resource-intensive
- Provides a higher level of security than symmetric encryption

Asymmetric Algorithms		
Keyword	Algorithm	Key Length (Bits)
RSA_2048	RSA	2048
RSA_1024	RSA	1024
RSA_512	RSA	512

Asymmetric key pair is made up of a private key and the public key. Each key can decrypt data encrypted by the other key pair. Asymmetric encryption/decryption are relatively resource-intensive, but they provide a higher level of security than symmetric encryption.

Asymmetric Algorithms		
Keyword	**Algorithm**	**Key Length (Bits)**
RSA_2048	RSA	2048
RSA_1024	RSA	1024
RSA_512	RSA	512

Certificates

A public key certificate is a digitally-signed statement that connects data of a public key to the identity of the person, device, or service that holds private key. Certificates are issued and signed by a Certification Authority (CA). SQL Server can also create a certificate and use it in the encryption process.

Creating Asymmetric Keys

- By default, the private key is protected by the DMK
- If no DMK has been created, a password is required to protect the private key.
- The private key can be 512, 1024, or 2048 bits long

```
CREATE ASYMMETRIC KEY StrongKey
WITH ALGORITHM = RSA_2048
ENCRYPTION BY PASSWORD = 'Some3xtr4Passw00rd';
GO
```

An asymmetric key is a database object protected at the database level. When executed without the FROM clause, CREATE ASYMMETRIC KEY generates a new key pair. When executed with the FROM clause, CREATE ASYMMETRIC KEY imports a key pair from a file or imports a public key from an assembly.

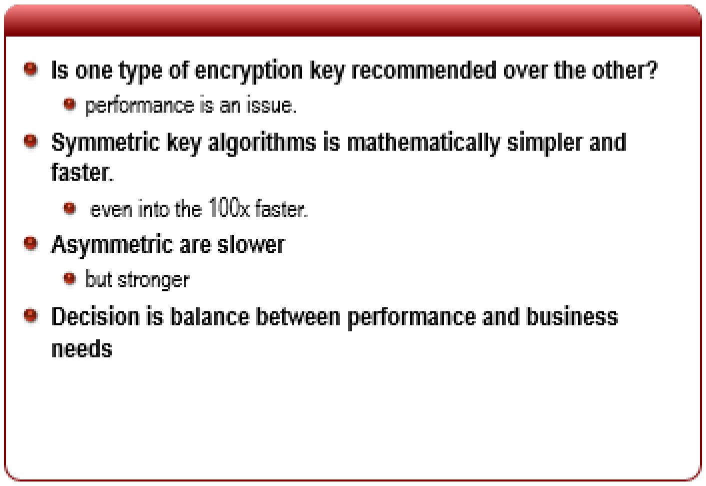
By default, the private key is protected by the DMK. If no DMK has been created, a password is required to protect the private key. If a DMK does exist, the password is optional.

The private key can be 512, 1024, or 2048 bits long.

```
CREATE ASYMMETRIC KEY StrongKey
      WITH ALGORITHM = RSA_2048
      ENCRYPTION BY PASSWORD = 'Some3xtr4Passw00rd';
GO
```

```
CREATE ASYMMETRIC KEY KeyPairFromFile  AUTHORIZATION Denis
      FROM FILE = 'C:\Keys\Asymmetric\DenisCert.tmp'
      ENCRYPTION BY PASSWORD = 'Some3xtr4Passw00rd';
GO
```

Implementing Asymmetric Encryption

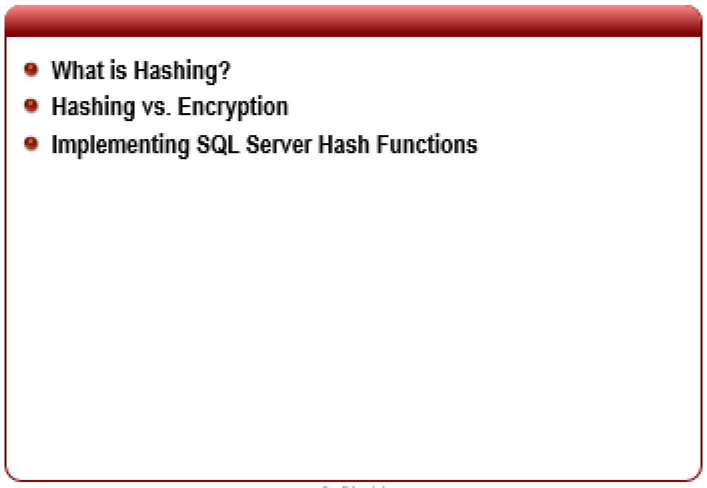


- Is one type of encryption key recommended over the other?
  - performance is an issue.
- Symmetric key algorithms is mathematically simpler and faster.
  - even into the 100x faster.
- Asymmetric are slower
  - but stronger
- Decision is balance between performance and business needs

Is one type of encryption key recommended over the other? Yes, but as always, performance is an issue. Symmetric key algorithms are mathematically simpler, and as a result, faster. The difference in speed can be significant even into the 100x faster range. Therefore, symmetric key algorithms are the way to go when encrypting data.

We can see this difference quite clearly with a simple example. We can set up two tables and create two keys. One set will be for an asymmetric key algorithm. The other will be for a symmetric key algorithm. We can then run through a number of rows of data and determine how much time it takes between the two algorithms. What we should see is that the symmetric key encryption is performed at a noticeably faster rate.

Lesson 6: Hashing



- What is Hashing?
- Hashing vs. Encryption
- Implementing SQL Server Hash Functions

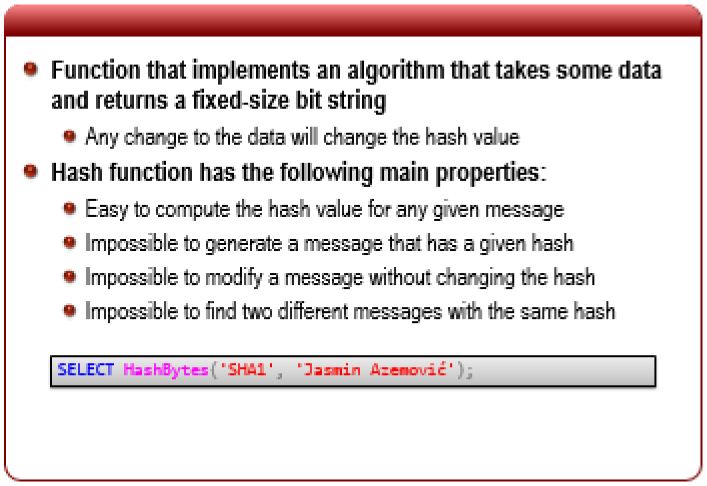
Sometimes business requirements will demand to hide some data without using encryption/decryption. There are plenty of mechanisms to make some data/information unreadable. Hashing is one of those mechanisms. Hash functions are powerful, fast and efficient ways to hide data and to check data integrity.

Objectives

After completing this lesson, you will be able to:

- Understand hashing
- Distinguish hashing from encryption
- Implement the hash function on SQL Server

What is Hashing?



- Function that implements an algorithm that takes some data and returns a fixed-size bit string
  - Any change to the data will change the hash value
- Hash function has the following main properties:
  - Easy to compute the hash value for any given message
  - Impossible to generate a message that has a given hash
  - Impossible to modify a message without changing the hash
  - Impossible to find two different messages with the same hash

```sql
SELECT HashBytes('SHA1', 'Jasmin Azemović');
```

A cryptographic hash function is a function that implements an algorithm that takes some data and returns a fixed-size bit string (the cryptographic hash value) such that any change to the data will change the hash value.

The hash function has the following main properties:

- Easy to compute the hash value for any given message
- Impossible to generate a message that has a given hash
- Impossible to modify a message without changing the hash
- Impossible to find two different messages with the same hash.

Cryptographic hash functions can also be used as ordinary hash functions to index data in hash tables, for fingerprinting, to detect duplicate data or uniquely identify files, and as checksums to detect accidental data corruption.

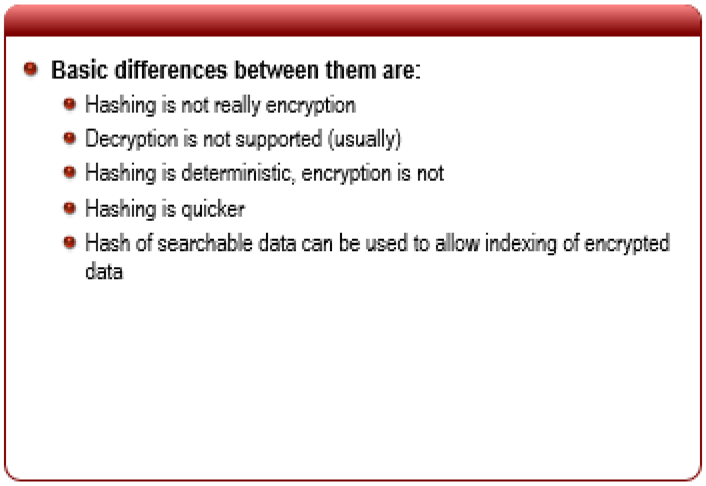
SQL Server uses the HASHBYTES functions. There are other implementations using .NET/CLR that you can include. Such as CHECKSUM() or BINARY_CHECKSUM() which can also be used.

```sql
      SELECT HashBytes('SHA1', 'Jasmin Azemović');
```

```
Result is:
------------------------------------------
0x026A71290A685DA3E09A11CA110014D3E0118191

(1 row(s) affected)
```

Hashing vs. Encryption



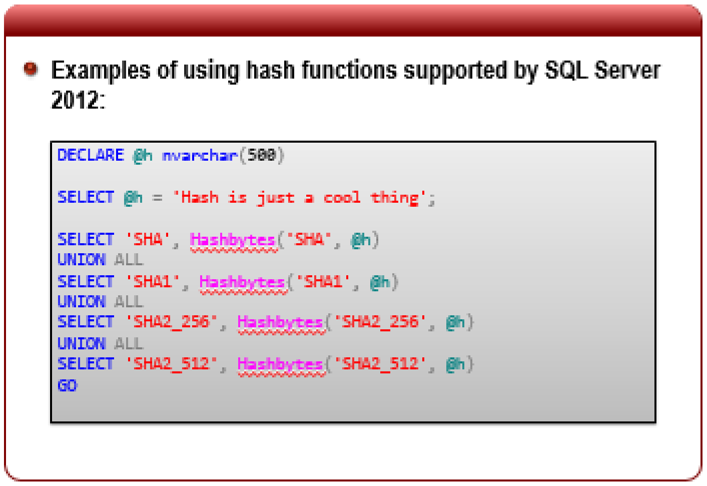
- Basic differences between them are:
  - Hashing is not really encryption
  - Decryption is not supported (usually)
  - Hashing is deterministic, encryption is not
  - Hashing is quicker
  - Hash of searchable data can be used to allow indexing of encrypted data

Hashing and encryption are not the same function and each have a totally different usage in practice. But, for the sake of clarity, we will note the basic differences between them:

- Hashing is not really encryption
- Decryption is not supported (usually)
- Hashing is deterministic, encryption is not
- Hashing is quicker
- Hash of searchable data can be used to allow indexing of encrypted data, because we cannot search encrypted data.
  - Only hash the portion of the encrypted data needed for searching, e.g. last four digits of a credit card number

The basic point to remember is that hashing is used when there is no need to make a reversible operation, whereas encryption is used when you need to decrypt data at some later point in time.

Implementing SQL Server Hash Functions



- Examples of using hash functions supported by SQL Server 2012:

```
DECLARE @h nvarchar(500)

SELECT @h = 'Hash is just a cool thing';

SELECT 'SHA', Hashbytes('SHA', @h)
UNION ALL
SELECT 'SHA1', Hashbytes('SHA1', @h)
UNION ALL
SELECT 'SHA2_256', Hashbytes('SHA2_256', @h)
UNION ALL
SELECT 'SHA2_512', Hashbytes('SHA2_512', @h)
GO
```

Below, you will find examples of using hash functions supported by SQL Server 2012:

```
DECLARE @h nvarchar(500)

SELECT @h = 'Hash is just a cool thing';

SELECT 'SHA', Hashbytes('SHA', @h)
UNION ALL
SELECT 'SHA1', Hashbytes('SHA1', @h)
UNION ALL
SELECT 'SHA2_256', Hashbytes('SHA2_256', @h)
UNION ALL
SELECT 'SHA2_512', Hashbytes('SHA2_512', @h)
GO
```

```
Result is:
------------------------------------------

SHA      0x4F8690AA5B77A33F7228652C98B5E437EA682F24
SHA1     0x4F8690AA5B77A33F7228652C98B5E437EA682F24
SHA2_256 0xE2871929C11B6D684AE32B527EF959570AB78B007800F7D01623D50864CA7216
SHA2_512
0x4C45D3DD20575DD8C58CA717B0E466847EAA470A72635FFCF631857959CB23CDADF4D6A37C964768B
B198C0E4B118B2495EDDA837CC6FE4A3624AD08B2DAE14C

(4 row(s) affected)
```

Practice: Implementing SQL Server Hash Functions

In this practice, you will implement a simple prototype for a user/password environment:

- Implementing Hash Functions

This practice requires SQL Server 2012 Enterprise Edition in Workgroup, not domain environment.

In this practice, you will implement a simple prototype for a user/password environment.

This practice requires SQL Server 2012 Enterprise Edition in Workgroup, not domain environment.

Exercise 1: Implementing Hash Functions

In this exercise, you will create set of database objects which implement basic password checking through the SQL Server hash function.

1. Generate all necessary database objects:

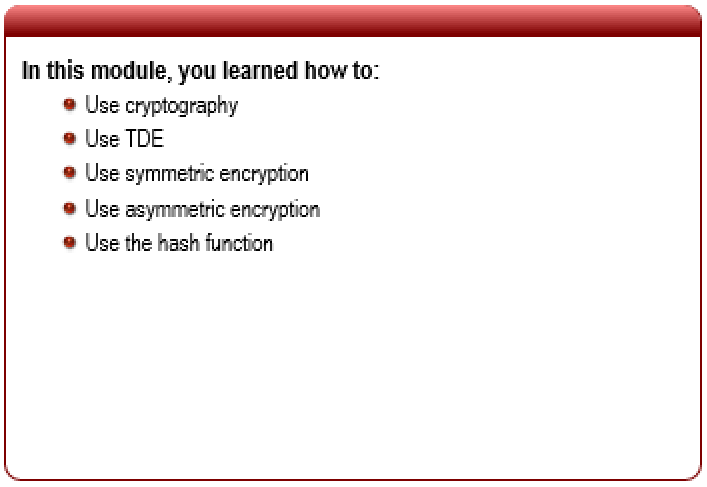
```
CREATE TABLE Users
( Name varchar(50)
, Password varbinary(max)
);
GO
-- Add users
INSERT Users SELECT 'Imran',  HASHBYTES('SHA2_512',
'Some3xtr4Passw00rd');
INSERT Users SELECT 'Selver', HASHBYTES('SHA2_512',
'SomeR3aly3xtr4Passw00rd');
GO

-- Stored procedure for validation
CREATE PROCEDURE Validate
   @Name varchar(200)
 , @Password varchar(200)
AS
if HASHBYTES('SHA2_512', @Password) = (SELECT Password
                                 FROM User
                                 WHERE Name = @Name
```

```
                                                                        )
    SELECT 'Password OK'
  ELSE
    SELECT 'Error'
    ;
  RETURN
  GO

  --Validation
  EXEC Validate 'Selver', 'Some3xtr4Passw00rd';
  GO
```

Summary


In this module, you learned how to:

- Use cryptography
- Use TDE
- Use symmetric encryption
- Use asymmetric encryption
- Use the hash function

As you have seen so far, protecting data is the most important thing in database environments. When all security elements fail (i.e. installation errors, authentication, authorization, bad access policy, etc.), there is no more protection. This module taught you how to implement advanced techniques for protecting data such as cryptography.

Objectives

After completing this module, you learned:

- Why cryptography is important
- How to use TDE
- How to use symmetric encryption
- How to use asymmetric encryption
- How to use the hash function