INFO-H-415 Project Overview- Security Database and SQL Server

Kirubel Yaekob Yasmine Daoud

December 2017

1 Introduction

A defense-in-depth strategy, with overlapping layers of security, is the best way to counter security threats. SQL Server provides a security architecture that is designed to allow database administrators and developers to create secure database applications and counter threats. Each version of SQL Server has improved on previous versions of SQL Server with the introduction of new features and functionality. However, security does not ship in the box. Each application is unique in its security requirements.

2 The principles of this project

- Server and Database users and roles in SQL Server
- Permission Hierarchy.
- Authorization and Permissions in SQL Server.
- Encryption hierarchy.
- Encryption Keys.
- Transact SQL Functions.
- Transparent Data Encryption in SQL Server (TDE).
- Requirements for TDE.
- Benefits and Disadvantage of TDE.

3 Server and Database users and roles in SQL Server

A SQL Server instance contains a hierarchical collection of entities, starting with the server. Each server contains multiple databases, and each database contains a collection of securable objects. Every SQL Server securable has associated permissions that can be granted to a principal, which is an individual, group or process granted access to SQL Server. The SQL Server security framework manages access to securable entities through authentication and authorization.

Some Terminologies

- **Securable** is a resource that someone might want to access (like the Financial Folder).
- A **Principal** is anything that might want to gain access to the securable (like Tom).
- A **Permission** is the level of access a principal has to a securable (like Read).
- **Authentication** is the process of logging on to SQL Server by which a principal requests access by submitting credentials that the server evaluates. Authentication establishes the identity of the user or process being authenticated.
- **Authorization** is the process of determining which securable resources a principal can access, and which operations are allowed for those resources.

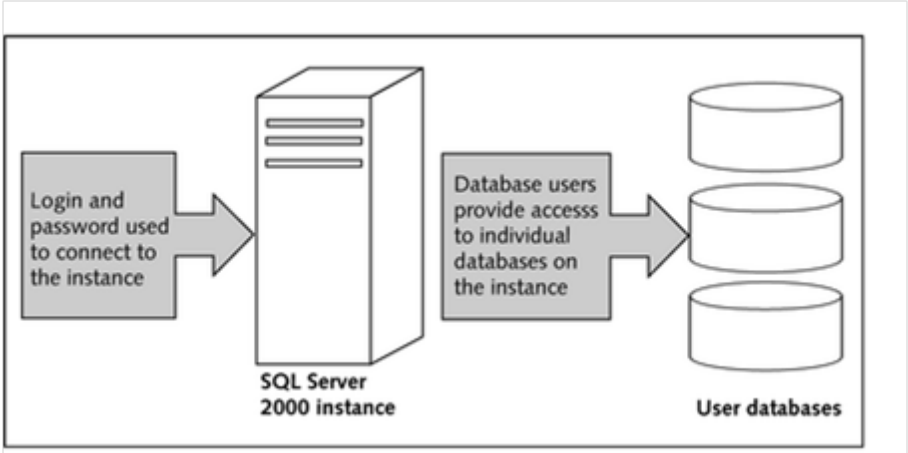
Security Authentication Modes

SQL Server supports two authentication modes, Windows authentication mode and mixed mode.

- **Windows authentication** is the default, and is often referred to as integrated security because this SQL Server security model is tightly integrated with Windows. Specific Windows user and group accounts are trusted to log in to SQL Server. Windows users who have already been authenticated do not have to present additional credentials. Along with limiting administration of a user to a single location (within a Windows domain), Windows authentication provides the following benefits:
  - Secure validation of credentials through Windows and encryption of passwords passed over the network
  - Windows password requirements
  - Automatic locking out of accounts that repeatedly fail to connect
  - Native auditing capabilities of Windows accounts

- **Mixed mode** supports authentication both by Windows and by SQL Server. User name and password pairs are maintained within SQL Server.
  - Mixed Authentication Mode is provided for backward compatibility with older applications designed to utilize SQL Server-based logins
  - Mixed Authentication Mode is also necessary for situations where users connecting to an instance of SQL Server do not have a Windows domain account

Logins
Authentication is implemented using logins in SQL Server. Logins are SQL Server object that provides connection access to an instance of SQL Server (authentication). Logins can be based on Windows users and groups defined natively in SQL Server



Login Types
SQL Server supports three types of logins:

- A local Windows user account or trusted domain account. SQL Server relies on Windows to authenticate the Windows user accounts.
- Windows group. Granting access to a Windows group grants access to all Windows user logins that are members of the group.
- SQL Server login. SQL Server stores both the username and a hash of the password in the master database, by using internal authentication methods to verify login attempts.

Roles in SQL Server

All versions of SQL Server use role-based security, which allows you to assign permissions to a role, or group of users, instead of to individual users. Fixed server and fixed database roles have a fixed set of permissions assigned to them.

- **Fixed Server Roles** have a fixed set of permissions and server-wide scope. They are intended for use in administering SQL Server and the permissions assigned to them cannot be changed. Logins can be assigned to fixed server roles without having a user account in a database.
- **Fixed Database Roles** have a pre-defined set of permissions that are designed to allow you to easily manage groups of permissions. There are several predefined roles in each database that provide sets of permissions for the database users who belongs to them.

Fixed Server Role	Description
Sysadmin	Allowed to perform any action in SQL Server
Serveradmin	Allowed to configure instance-wide settings and shut down the instance
Setupadmin	Allowed to manage linked servers and startup procedures
Securityadmin	Allowed to manage logins and provide CREATE DATABASE permissions to them, read error logs, and change passwords
Processadmin	Allowed to manage running processes in SQL Server
Dbcreator	Allowed to create, modify, and delete databases
Diskadmin	Allowed to manage disk files
Bulkadmin	Allowed to use the BULD INSERT statement to perform mass imports of data

Fixed Database Role	Description
db_owner	Allowed to perform all of the operations permitted to the other roles, as well as activities to maintain and configure the database
db_accessadmin	Allowed to manage database users mapped from) Windows users, Windows Groups, and SQL) Server Logins)
db_datareader	Allowed to see (read access) all data in all of the user-defined tables in a database
db_datawriter	Allowed to insert, update, and delete data from all user-defined tables

Fixed Database Role	Description
db_ddladmin	Allowed to create, modify, and remove all database objects, like tables and views
db_securityadmin	Allowed to manage roles and role membership, as well as to apply permissions to database users and roles
db_backupoperator	Allowed to back up the database
db_denydatereader	Not allowed to view data in the database
db-denydatawriter	Not allowed to modify data in the database
public	The default role of which every database user is a member. If a user does not have permission to access an object like a table, then the permissions of the public role are checked as last resort

Database Users

Database users are individual accounts stored within each database that control access to database objects through permissions.Database users are mapped to Windows users, Windows groups and SQL Server logins to grant access to the database.

DBO and the Guest Database User
Every database in an instance of SQL Server of recent versions can have two special users:

- **Database owner:** Special user that has permissions to perform all database activities
- **Guest user:** Special account that allows database access to a login without a mapped database user

Example Creating a Login with Password

```
USE [master]
GO
CREATE LOGIN [BUSH] WITH PASSWORD=N'password1', DEFAULT_DATABASE=[AdventureWorksDW2012],
 CHECK_EXPIRATION=OFF, CHECK_POLICY=OFF
GO
```

Example Creating a user named "BUSH" mapped to login "BUSH" created above to access database 'AdventureWorksDW2012'

```
     USE [AdventureWorksDW2012]
GO
CREATE USER [BUSH] FOR LOGIN [BUSH]
GO
```

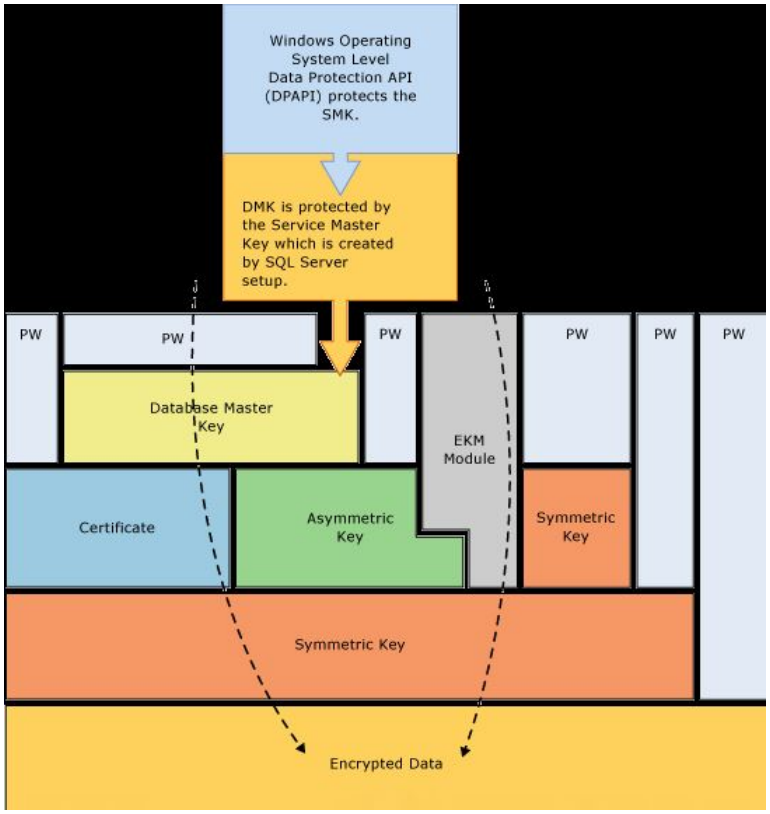
Example Creating a user named "OBAMA" to access database 'AdventureWorksDW2012'. This Can only be used in a contained database.

```
     USE [AdventureWorksDW2012]
GO
CREATE USER [OBAMA] WITH PASSWORD=N'password1'
GO
```

4 Encryption Hierarchy

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

The following illustration shows that each layer of the encryption hierarchy encrypts the layer beneath it, and displays the most common encryption configurations. The access to the start of the hierarchy is usually protected by a password.



5 Encryption Keys

- Asymmetric Keys
  An asymmetric key is made up of a private key and the corresponding public key. Each key can decrypt data encrypted by the other. Asymmetric encryption and decryption are relatively resource-intensive, but they provide a higher level of security than symmetric encryption. An asymmetric key can be used to encrypt a symmetric key for storage in a database. It can be DES and AES keys.

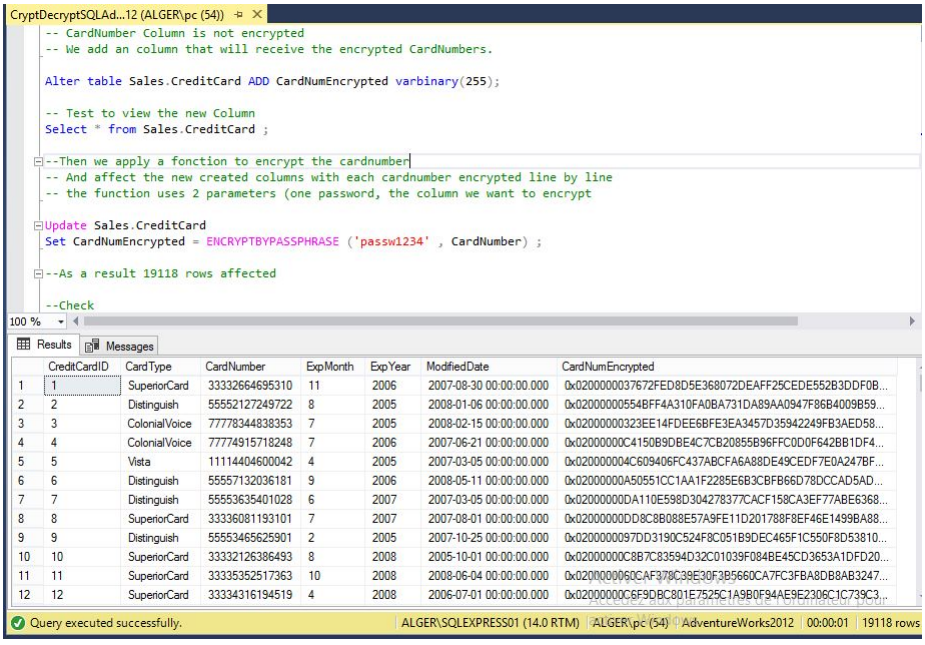
- Symmetric Keys

  A symmetric key is one key that is used for both encryption and decryption. Encryption and decryption by using a symmetric key is fast, and suitable for routine use with sensitive data in the database. it can be RSA.

6 Transact SQL-Functions

Individual items can be encrypted as they are inserted or updated using Transact-SQL functions. Encrypt data with a passphrase using the TRIPLE DES algorithm with a 128 key bit length.

Here is an example on the database AdventureWork2012. We applied the function to encrypt the Credit Card Numbers. It could be found in the file CryptDecryptSQLAventure.sql



7 Transparent Data Encryption in SQL Server (TDE).

Transparent Data Encryption (often abbreviated to TDE) is a technology employed by Microsoft, IBM and Oracle to encrypt database files. TDE offers encryption at file level. TDE solves the problem of protecting data at rest, encrypting databases both on the hard drive and consequently on backup media. It does not protect data in transit nor data in use. Enterprises typically employ TDE to solve compliance issues such as PCI DSS which require the protection of data at rest.

The TDE requires the following:

- SQL Server 2008/2012/2014 Entreprise Edition.
- Master Database - Master Key.
- Master Database - Certificate.
- User Database - Database Encryption Key.
- Implemented with a simple ALTER DATABASE COMMAND:
  ALTER DATABASE [Database_Name] SET ENCRYPTION ON GO

8 Benefits and disadvantage of TDE

The benefits of TDE are:

- Fairly simple to implement.
- No changes to the application tier required.
- Is invisible to the user.

- Works with high availability features, such as mirroring, AlwaysOn and log shipping.
- Works with older versions of SQL Server, back to 2008.

The Disadvantages of TDE are :

  - Only encrypts data at rest, so data in motion or held within an application is not encrypted.
  - All data in the database is encrypted – not just the sensitive data.
  - Requires the more expensive Enterprise Edition (or Developer or DataCenter Edition) of SQL Server.
  - The amount of compression achieved with compressed backups will be significantly reduced.
  - There is a small performance impact.
  - FileStream data is not encrypted.
  - Some DBA tasks require extra complexity, for instance restoring a backup onto another server.
  - As TempDB is encrypted, there is potentially an impact on non-encrypted databases on the same server.
  - The master database, which contains various metadata, user data and server level information is not encrypted.

9 Conclusion

For completeness TDE is not the only database encryption technique available within SQL Server, some of the others are:

-The business logic within individual stored procedures can be encrypted using the 'ENCRYPTION' keyword.

-Individual data items (i.e. column or cell-level encryption) can be encrypted and decrypted using the 'ENCRYPTBYPASSPHRASE' and 'DECRYPTBYPASSPHRASE' statement along with a pass phrase. ENCRYPTBYKEY/DECRYPTBYKEY and ENCRYPTBYCERT/DECRYPTBYCERT are similar but use a key or certificate to encrypt the data.

-Data items can also be encrypted and decrypted using .net CLR functions.

Some third party software such as Redgate SQL Backup include backup file encryption.