

Protect data in-transit and at rest

1. Introduction

<https://learn.microsoft.com/en-us/training/modules/protect-data-transit-rest/1-introduction>

Introduction

Completed

Explore the encryption options available within Microsoft SQL Server, Azure SQL Database, and Azure SQL Managed Instance. Each of the various platforms supports different database encryption options. Students explore these data encryption options and how to configure them.

Consider the following three scenarios when evaluating data encryption methods.

Scenario	Definition
Data at rest	Encrypting it while it's on file storage.
Data in transit	Encrypting it while it travels through private or public network communication channels.
Data in use	Encrypting it while it's in RAM or CPU caches.

2. Explore Transparent Data Encryption

<https://learn.microsoft.com/en-us/training/modules/protect-data-transit-rest/2-explore-transparent-data-encryption>

Explore Transparent Data Encryption

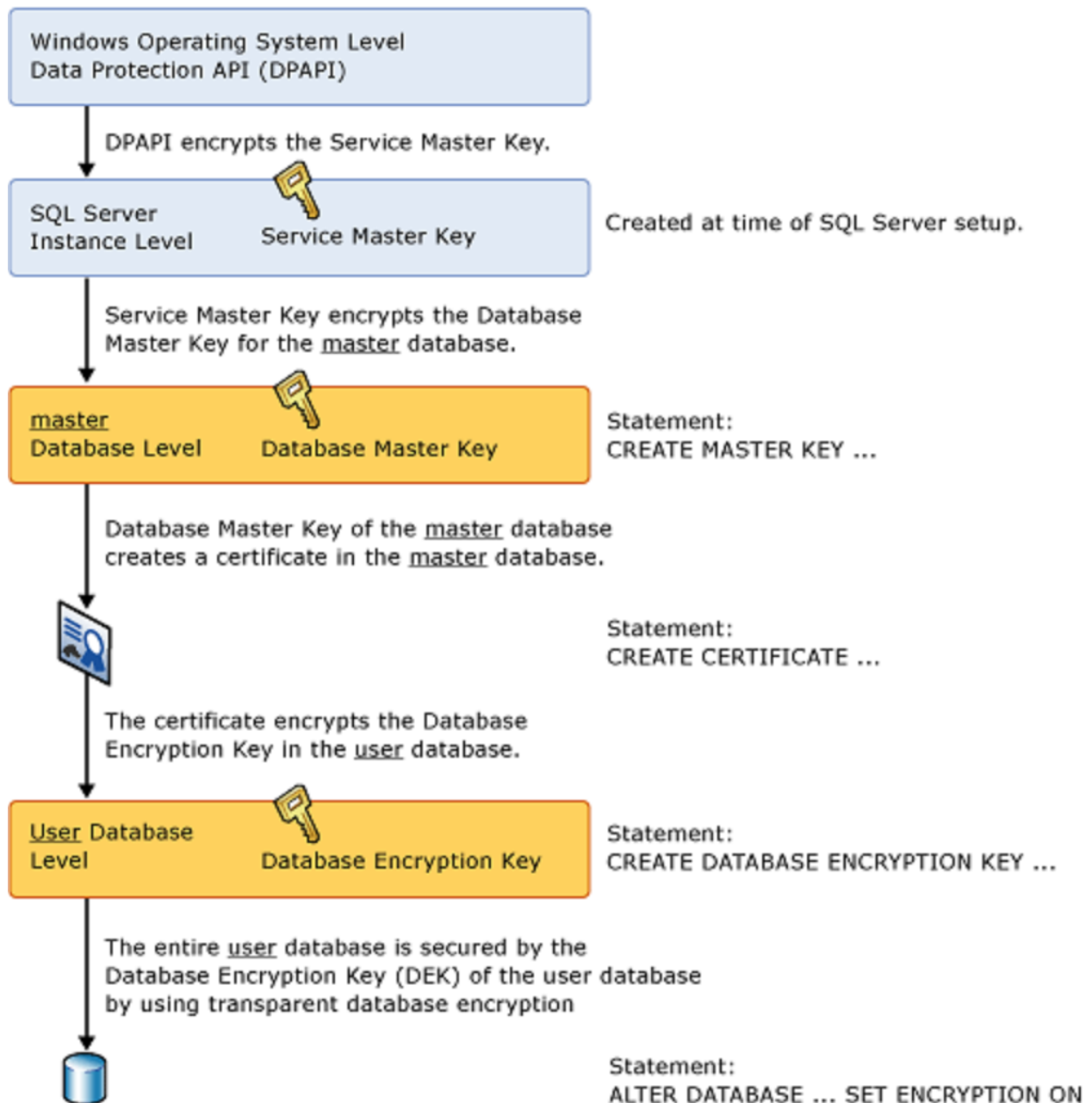
Completed

Microsoft SQL Server's Transparent Data Encryption (TDE) encrypts all data within a target database at the page level. Data is encrypted as it's written to the data page on disk and decrypted when read into memory, resulting in all data pages on disk being encrypted.

TDE doesn't encrypt data at the table or column level. Anyone with the appropriate permissions can read, copy, and share the data. Encryption at rest protects against restoring a backup to an unsecured server or copying database and transaction log files to another unsecured server. No decryption occurs during the backup operation.

TDE protects data at rest and complies with various industry laws, regulations, and guidelines. It allows software developers to encrypt data using AES and 3DES encryption algorithms without changing existing applications.

Transparent Database Encryption Architecture



Databases created in Azure SQL Database after May 2017 have TDE enabled automatically. Databases created before May 2017 need TDE to be manually enabled. For Azure SQL Managed Instance, TDE is

enabled by default for databases created after February 2019. Databases created before February 2019 need TDE to be manually enabled.

To enable TDE in an Azure SQL Database, edit the database in the Azure portal. From the **Transparent data encryption** pane, select to enable data encryption.

 Save  Discard  Feedback

Transparent data encryption encrypts your databases, backups, and logs at rest without any changes to your application. To enable encryption, go to each database. [Learn more](#)

Transparent data encryption ⓘ ☒ Service-managed key
☐ Customer-managed key

By default, databases in Azure SQL Database are encrypted using a Microsoft-provided certificate (service-managed key). Azure also offers a Bring Your Own Key (BYOK) option, allowing you to use a customer-managed key created by your company and uploaded to Azure Key Vault. If the customer-managed key is removed from Azure, database connections are closed, and access to the database are denied.

Enabling TDE within a Microsoft SQL Server database is an easy process as only a few T-SQL commands are required. This process involves the following steps:

1. Set a master key within the master database using the `CREATE MASTER KEY ENCRYPTION` command.
2. Create a certificate in the master database using the `CREATE CERTIFICATE` command.
3. Create a database encryption key within the database using the `CREATE DATABASE ENCRYPTION KEY` command.
4. Enable the encryption key using the `ALTER DATABASE` command.

```
USE master;
GO

CREATE MASTER KEY ENCRYPTION BY PASSWORD = '<your-pwd>';
GO

CREATE CERTIFICATE MyServerCert
    WITH SUBJECT = 'TDEDemo_Certificate';
GO

USE [TDE_Demo];
GO

CREATE DATABASE ENCRYPTION KEY
    WITH ALGORITHM = AES_256 ENCRYPTION BY SERVER CERTIFICATE MyServerCert;
GO

ALTER DATABASE TDE_Demo SET ENCRYPTION ON;
GO
```

Once TDE is enabled, it takes time to encrypt the database as each page must be read, encrypted, and written back to disk. The larger the database, the longer this process takes. This background process runs at a low priority to avoid overloading the system's I/O or CPU.

The certificate used by TDE must be manually backed up and stored securely. SQL Server integrates with Enterprise Key Managers (EKMs) like Azure Key Vault to manage encryption keys. Managing the certificate is crucial because if it's lost and the database needs to be restored from a backup, the restore fails as the database can't be read.

Note

To use TDE with databases in an Always On Availability Group, the certificate used to encrypt the database must be backed up and restored to the other servers within the Availability Group that will be hosting copies of the database.

Customer-managed keys

You can alternately use BYOK and take advantage of an Azure key vault. The advantages of using customer-managed keys are:

- Full and granular control over usage and management of the TDE protector
- Transparency of the TDE protector usage
- Ability to implement separation of duties in the management of keys and data within the organization
- The key vault administrator can revoke key access permissions to make encrypted database inaccessible
- Central management of keys in AKV
- Greater trust from your end customers because AKV is designed so that Microsoft can't see or extract encryption keys

You can also take advantage of using a [user-assigned managed identity \(UMI\)](#) with customer-managed keys for TDE, which:

- Enables the ability to preauthorize key vault access for Azure SQL logical servers by creating a user-assigned managed identity and granting it access to key vault, even before the server or database are created.
- Allows creation of an Azure SQL logical server with TDE and CMK enabled.
- Enables the same user-assigned managed identity to be assigned to multiple servers, eliminating the need to individually turn on system-assigned managed identity for each Azure SQL logical server and providing it access to key vault.
- Provides the capability to enforce CMK at server creation time with an available built-in Azure policy.

[Automatic key rotation](#) are introduced for customer-managed keys using TDE. When enabled, the server continuously checks the key vault for any new versions of the key being used as the TDE

protector. If a new version of the key is detected, the TDE protector on the server is automatically rotated to the latest key version within 60 minutes.

Azure disk encryption

In addition to these SQL Server security features, Azure VMs include an extra layer of security, Azure Disk Encryption—a feature that helps protect and safeguard data and meet organization and compliance commitments. If you're using TDE, your data is protected by multiple layers of encryption with Azure Disk Encryption.

3. Configure server and database firewall rules

<https://learn.microsoft.com/en-us/training/modules/protect-data-transit-rest/3-configure-server-and-database-firewall>

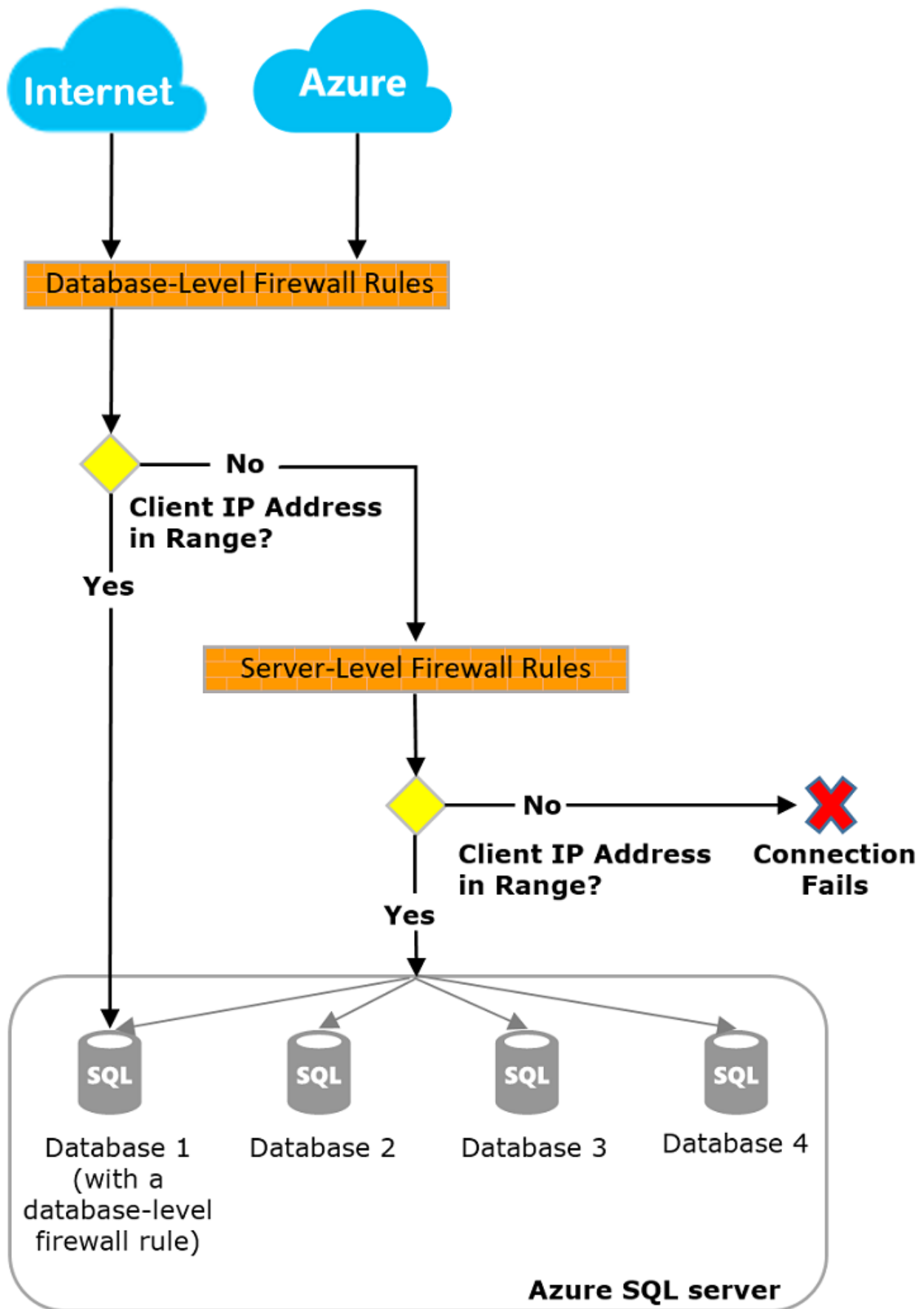
Configure server and database firewall rules

Completed

Firewalls prevent unauthorized users from accessing protected resources. Each Azure SQL Database maps to a public IP address hosted by Microsoft. In each Azure region, one or more public IP addresses allow you to reach your database gateway, which then directs you to your database.

How firewall works

As shown in the following diagram, connection attempts coming from the internet and Azure must go through the firewall before they reach your server or database.



Azure provides built-in firewalls to limit access in order to protect your database and your data. In Azure SQL Database there are two distinct sets of firewall rules: server-level firewall rules and database-level firewall rules.

Server-level firewall rules

Both server and database level firewalls use IP Address rules instead of SQL Server Logins, and allow all users at the same public IP Address to access the SQL Server. For most companies, this is their outbound IP address.

Server-level firewalls are configured to allow users to connect to all databases on the server. Database level firewalls are used to grant or block specific IP Addresses from accessing specific databases.

Server level firewall rules can be configured using the Azure portal or using the `sp_set_firewall_rule` stored procedure from within the *master* database.

Note

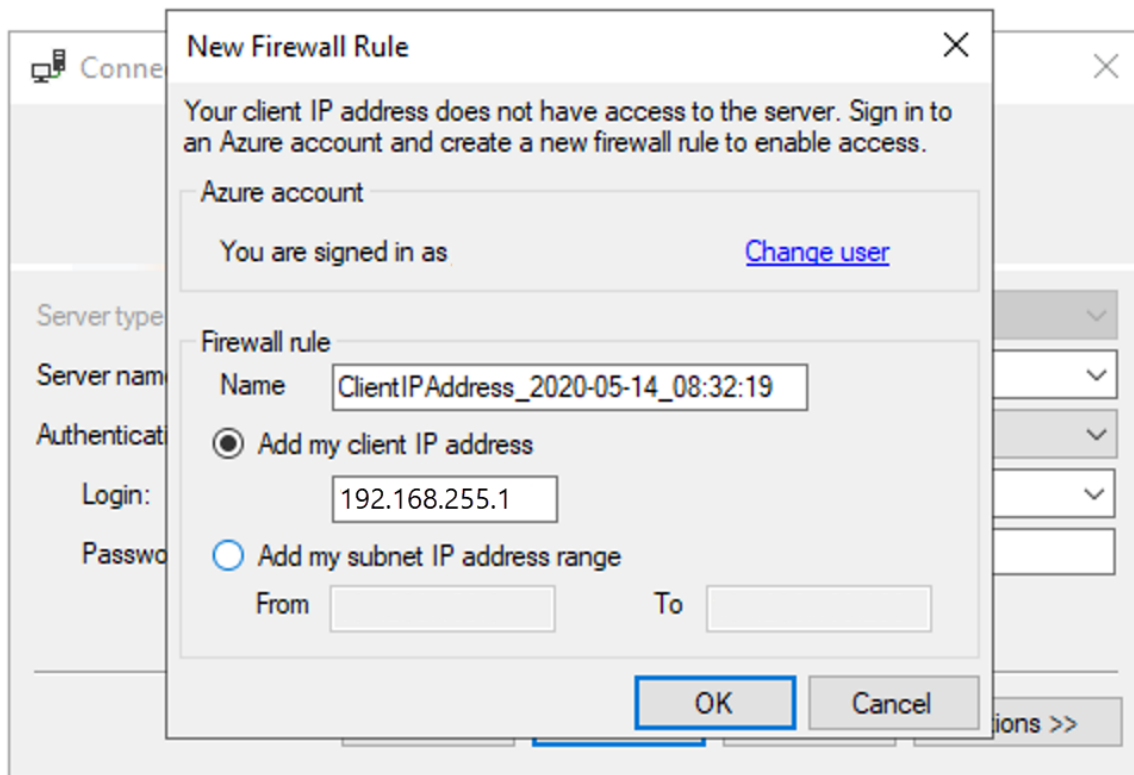
The **Allow Azure Services and resources to access this server** server setting counts as a single firewall rule when enabled.

Database-level firewall rules

Database-level firewall rules are configured through T-SQL only using the `sp_set_database_firewall_rule` stored procedure from within the user database.

Upon connection, Azure SQL Database first checks for a database-level firewall rule corresponding to the database name specified in the connection string. If no such rule exists, the firewall then checks the server-level IP firewall rules. Server-level IP firewall rules apply to all databases on the server. If a matching rule is found at either level, the connection is established.

If neither exist and the user is connecting through SQL Server Management Studio, they'll be prompted to create a firewall rule.



Virtual network endpoints

Virtual network endpoints allow traffic from a specific Azure Virtual Network. These rules apply at the server level, not just the database level.

Additionally, the service endpoint applies to only one region, which is the underlying endpoint's region.

Another important consideration is that the virtual network connecting to the Azure SQL Database must have outbound access to the database's public IP address. This can be configured using service tags for Azure SQL Database.

Private link

The Private Link feature allows you to connect to Azure SQL Database and other PaaS offerings using a private endpoint.

A private endpoint allows for a connection to your Azure SQL Database to go entirely over the Azure backbone network and not over the public internet.

This feature provides a private IP address on your Virtual Network. Another feature of private link is that it allows for Azure Express Route connections through that circuit.

Private link offers several benefits including cross-region private connectivity and protection against data leakage by only allowing connections to specific resources.

4. Explain object encryption and secure enclaves

<https://learn.microsoft.com/en-us/training/modules/protect-data-transit-rest/4-explain-object-encryption-secure-enclaves>

Explain object encryption and secure enclaves

Completed

In addition to supporting encryption at rest, SQL Server supports encrypting data within columns using Always Encrypted. Once data is encrypted, the application accessing the database must have the correct certificate in order to view the plain text values of the data.

Always Encrypted

Always Encrypted allows for the encryption of data within the client application, protecting sensitive data from malware and high-privileged users, such as Database Administrators (DBAs), server admins, cloud admins, or those who manage the data but should have no access. This encryption happens automatically based on the settings within the Microsoft SQL Server database, which tell the application what the encryption settings on the database column are.

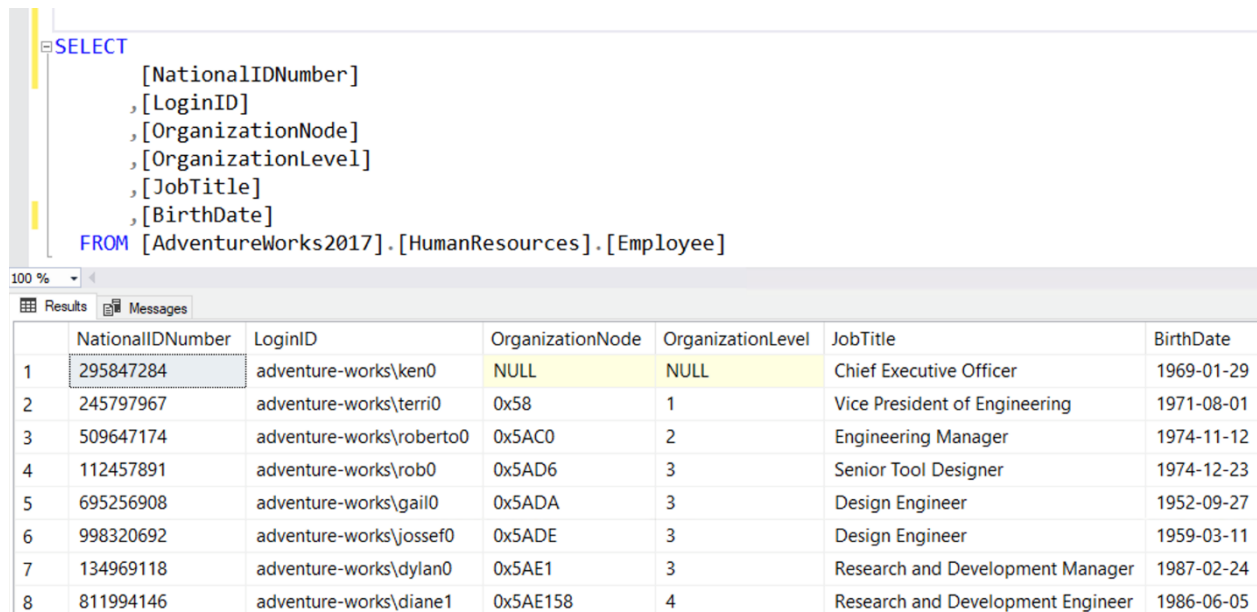
The following table provides some scenarios for Always Encrypted usage:

Scenario	Definition
Client and data on-premises	For scenarios where you need to protect your on-premises database from high-privileged users, for example, external vendors managing SQL Server.
Client on-premises with data in Azure	In this scenario, to ensure Microsoft cloud administrators have no access to the data, Always Encrypted keys are stored in key store hosted on-premises, for SQL Database or SQL Server running in a virtual machine on Microsoft Azure.
Client and Data in Azure	In this scenario, the environment is fully hosted on Azure. While Always Encrypted doesn't completely isolate data from cloud administrators, the customer still benefits from the fact that the data is encrypted in the database.

Always Encrypted is based on a master encryption key and a column encryption key. Having both keys allows each column to be encrypted using a different encryption key for maximum data

protection. Always Encrypted has various key stores that can store the certificate used by encryption.

Here's an example of enabling Always Encrypted. You can see that *NationalIDNumber* and *BirthDate* columns are both in plain text.

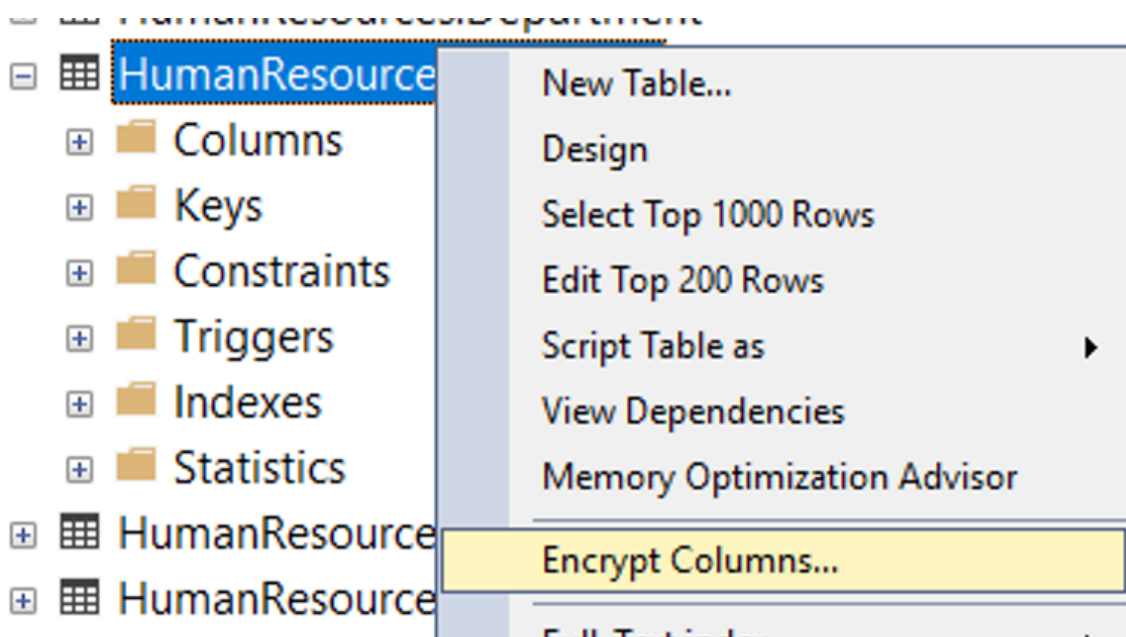


The screenshot shows a SQL query in the query editor and its results in the Results pane. The query selects columns from the Employee table in the HumanResources schema of the AdventureWorks2017 database. The results pane shows 8 rows of data. The first row has the first two columns highlighted.

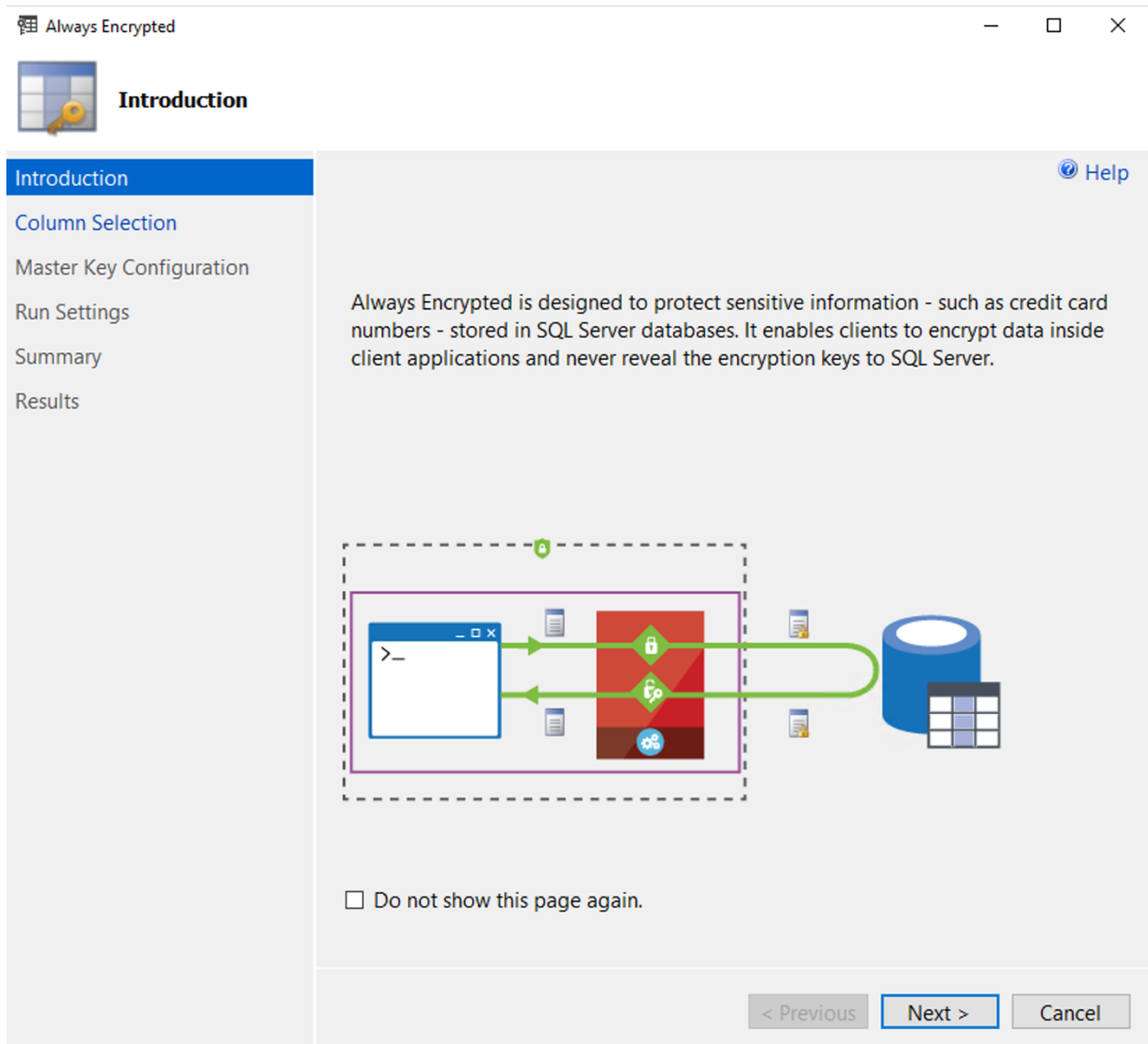
```
SELECT
    [NationalIDNumber]
  , [LoginID]
  , [OrganizationNode]
  , [OrganizationLevel]
  , [JobTitle]
  , [BirthDate]
FROM [AdventureWorks2017].[HumanResources].[Employee]
```

	NationalIDNumber	LoginID	OrganizationNode	OrganizationLevel	JobTitle	BirthDate
1	295847284	adventure-works\ken0	NULL	NULL	Chief Executive Officer	1969-01-29
2	245797967	adventure-works\terri0	0x58	1	Vice President of Engineering	1971-08-01
3	509647174	adventure-works\roberto0	0x5AC0	2	Engineering Manager	1974-11-12
4	112457891	adventure-works\rob0	0x5AD6	3	Senior Tool Designer	1974-12-23
5	695256908	adventure-works\gail0	0x5ADA	3	Design Engineer	1952-09-27
6	998320692	adventure-works\jossef0	0x5ADE	3	Design Engineer	1959-03-11
7	134969118	adventure-works\dylan0	0x5AE1	3	Research and Development Manager	1987-02-24
8	811994146	adventure-works\diane1	0x5AE158	4	Research and Development Engineer	1986-06-05

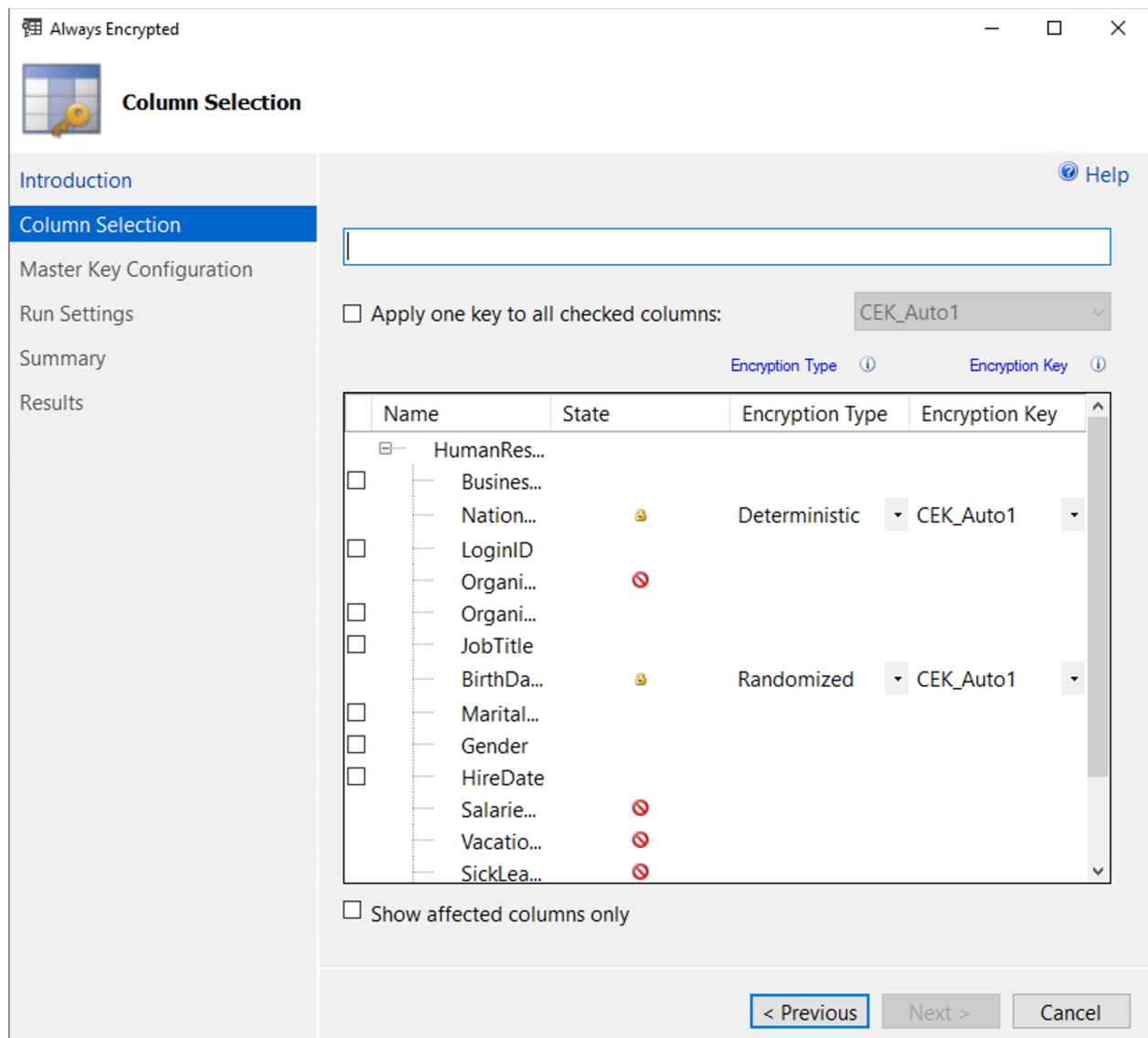
The next few images show you how we can encrypt both of these columns using Always Encrypted. The encryption could be done using T-SQL, but in this example, we use the wizard from SQL Server Management Studio. You can reach the wizard by right-clicking on the table name in Object Explorer as shown below.



When you select **Encrypt Columns...**, the wizard launches.



Select **Next** to choose the columns you want to encrypt.



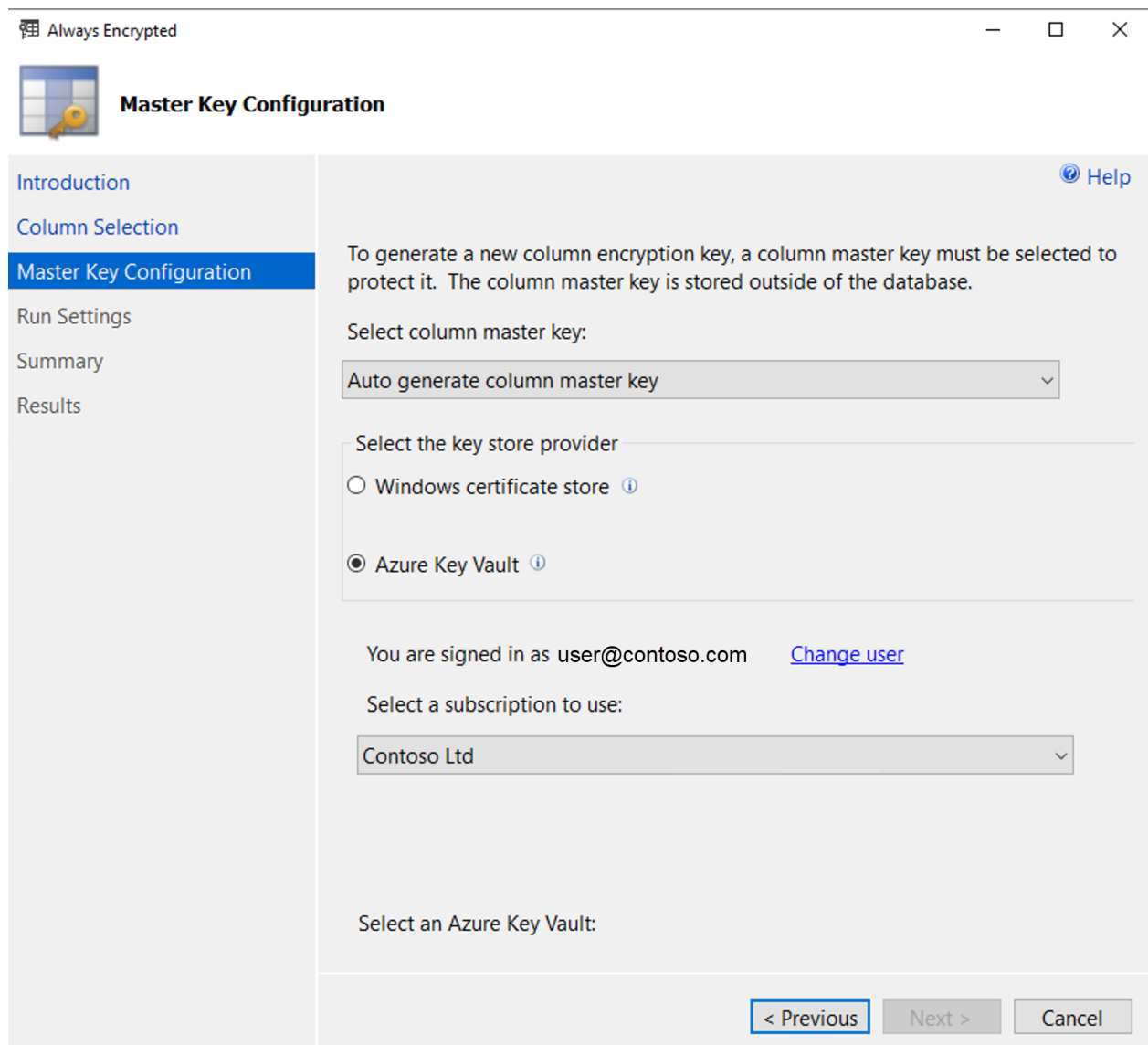
There are two different types of encryption specified. The *NationalIDNumber* column is encrypted with **Deterministic** encryption, and the *BirthDate* column is encrypted using **Randomized** encryption.

Randomized encryption is more secure than deterministic encryption but comes with limitations. Once a column is created, you can't change its encryption type. It's recommended to use randomized encryption for columns with a few well-known distinct values that could be guessed by someone with access to the encrypted data, such as a three-digit credit card verification code.

Always Encrypted with randomized encryption is limited because the same value is encrypted differently each time. This means you can only return these columns in your results. In contrast, deterministic encryption always encodes a value the same way, allowing for comparisons using equality and inequality operators, as well as joins, grouping, and indexing.

Also, the wizard generates a column encryption key, which performs the data encryption. Each encrypted column can have its own key, or you can use the same key for multiple columns.

After identifying the columns you're encrypting, you can select **Next** and you'll see the **Master Key Configuration** screen:



In this screen, you create the column master key, which is used to encrypt the column encryption keys. You can supply your own key, if you're using T-SQL to encrypt the columns. This key must be stored in a key store such as the Windows Certificate Store, Azure Key Vault, or a hardware security module. The database engine never stores the column master key, and only contains the metadata about where it's stored. Not storing the master key protects data access from users who have full access to the database.

For the highest level of security, the key should be stored within a third-party key store such as Azure Key Vault. Never generate the keys on the server hosting your database, as the key could potentially be extracted from memory on that server.

In the example, the key is being stored in Azure Key Vault. On the next screen, the wizard will provide you with the option to either finish the encryption process now, or to generate a PowerShell script. Once you complete the process, the data appears as encrypted to anyone querying the data without the key.

select * from HumanResources.Employee

Results	Messages	BusinessEntityID	NationalIDNumber	LoginID	OrganizationNode	OrganizationLevel	JobTitle	BirthDate
1		1	0x01C3979CCA373497E69A372F30A091E096D1A8C60C5738E...	adventure-works\ken0	NULL	NULL	Chief Executive Officer	0x01C395FEDE5C8965A81D512F3C5EA6E6D97AF13840AA6CF...
2		2	0x01B8850F9C9599EAE1B0653F01FCE7321671F6F0FA0822D3...	adventure-works\terri0	0x58	1	Vice President of Engineering	0x014DE4461EA605D07B955A82832523A57206970670C7D...
3		3	0x01BDD49E6DF84DFB95FD5C42EC511A86A6C5FCA4588C63F...	adventure-works\roberto0	0x5AC0	2	Engineering Manager	0x01CE6DAD98D6B28AF76EA565565615A662A025F161858CD...
4		4	0x01E8634C71B297A23C9AC247F682C28E00976CFA5265F836...	adventure-works\robo0	0x5AD6	3	Senior Tool Designer	0x0117B23F362D932E0380FAF28A9EBF7B59C68AA21840A8725...
5		5	0x01B49F37A7628AB9093EC44AA071FFAF3193B2F820A5B5C5...	adventure-works\gail0	0x5ADA	3	Design Engineer	0x01DFA48EB32386DA7F39C8BEA1CF34CCF71A2F38F3185519...
6		6	0x01D2760AEC79ED9659F70DD3AC8E26232D64584D3B2E1F...	adventure-works\josse0	0x5ADE	3	Design Engineer	0x0186FF5BAD106F4ACDFF47194FB0484615A7D812959B5D492...
7		7	0x019FD96C3C09F255FAA6F789D105CA8D41E49D3B972E0E3...	adventure-works\dylan0	0x5AE1	3	Research and Development Manager	0x0134262E465846A90A7B2E13A422092AC1A582829D3A19F38...
8		8	0x019280287AAAD3386A78AA9071FD9F8395C68B2538320F03...	adventure-works\diane1	0x5AE158	4	Research and Development Engineer	0x0153027957F29425CF8949093C8A087428128C871E018C6C1...
9		9	0x01B2A71B39816D3BE16023B774F92B8C72060A69E9E00E1E4...	adventure-works\gigi0	0x5AE168	4	Research and Development Engineer	0x01F458C1E6E5C0D493F13AD2617844C2BEDAE057E074E5713...
10		10	0x01975B145982F73ED3E8A4253D805CF0AA48DA3613083CC4...	adventure-works\michael6	0x5AE178	4	Research and Development Manager	0x01FA2B454898604F48B6F2985B4D9D022305E38CF4FA6B4C2...
11		11	0x01553EF331044852447480DA345E586285ACDF7FED163E25F...	adventure-works\ovidio0	0x5AE3	3	Senior Tool Designer	0x01EC391F0168F8B338026F380643B3CED587AA0CA23F65BA...
12		12	0x01041122F34858C6171448A3FBEEF2BA97EC148BC8E3E119E...	adventure-works\thierry0	0x5AE358	4	Tool Designer	0x01D946A0CF88619E8C2128B8C9ED95DDE8B7758DFD0467...
13		13	0x01C53D73AC0DFCE594D1CF72DE9837A0F11A9A317405862...	adventure-works\janice0	0x5AE368	4	Tool Designer	0x018D56101A359B7680CCD9E00FD13013C9473746CFD98C7...
14		14	0x015EADCC6AC88F6C12844545074303726F10A426DD95C90B...	adventure-works\michael8	0x5AE5	3	Senior Design Engineer	0x0101929CAC7B3F4A8A7F3FE1C729C034EBF582D5D357DF11...
15		15	0x0120328DDE84AC518B081396151CE1191F473EE04DF36AF1...	adventure-works\sharon0	0x5AE7	3	Design Engineer	0x018761024DFE8DE4CABDFAB0E0449A2707B0413E2786287...

In order to decrypt data from an Always Encrypted column, your application needs an Always Encrypted driver to connect to the database, followed by the following actions:

1. The application has access to the key store where the Always Encrypted keys are stored
2. The application then retrieves the data
3. Data that is written back to the database is encrypted at the client through the driver

In addition to the driver, the application's connection string needs to have the setting **Column Encryption Setting=enabled** provided. This setting causes a metadata lookup to be made for each column that is used by the application.

Note

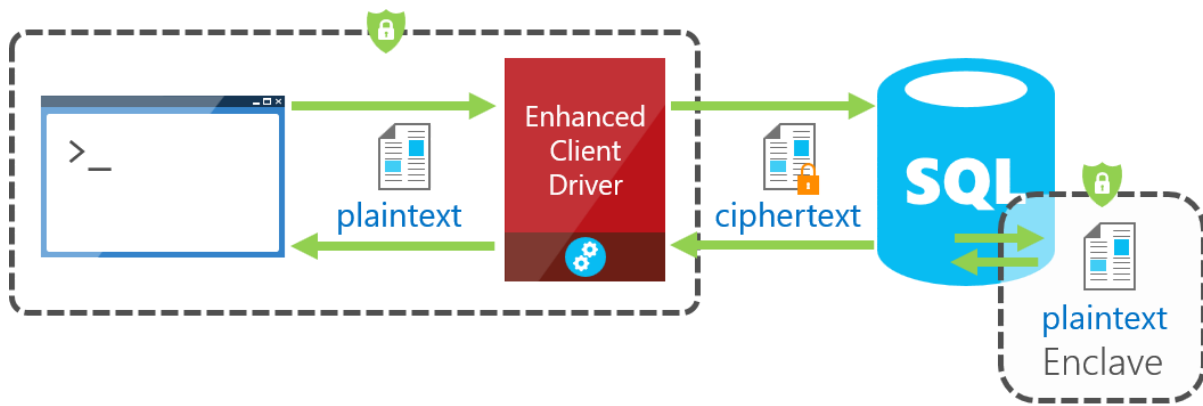
To minimize metadata lookups, the application needs to update the *SqlCommandColumnEncryptionSetting* on the *SqlConnection* objects within the .NET application. These settings must be set for each database query that the application submits.

Secure enclaves

Always Encrypted supports a feature called secure enclaves, which allows more robust querying of encrypted data.

A secure enclave is a secured region of memory within the SQL Server process that acts as a trusted execution environment for processing encrypted data. This enclave appears as a black box to SQL Server, and it isn't possible to view any data or code, even with a debugger.

The image shows the architecture of this process:



Always Encrypted with secure enclaves also addresses some of the limitations of Randomized encryption, which enables pattern matching, comparison operations, and indexing on columns using this encryption type.

5. Enable encrypted connections

<https://learn.microsoft.com/en-us/training/modules/protect-data-transit-rest/5-enable-encrypted-connections>

Enable encrypted connections

Completed

It's possible to encrypt data that is transmitted across a network between an instance of SQL Server and a client application with Transport Layer Security (TLS). The TLS encryption is performed within the protocol layer and is available to all supported SQL Server and Azure SQL database services.

Certificates

You must run SQL Server Configuration Manager with a local administrator account in order to install certificates for use by SQL Server.

Furthermore, the certificate must satisfy the following conditions:

- The certificate must be located in the local computer certificate store or the current user certificate store.
- The SQL Server service account must have permission to access the certificate.
- The certificate must be within a valid period.

Note

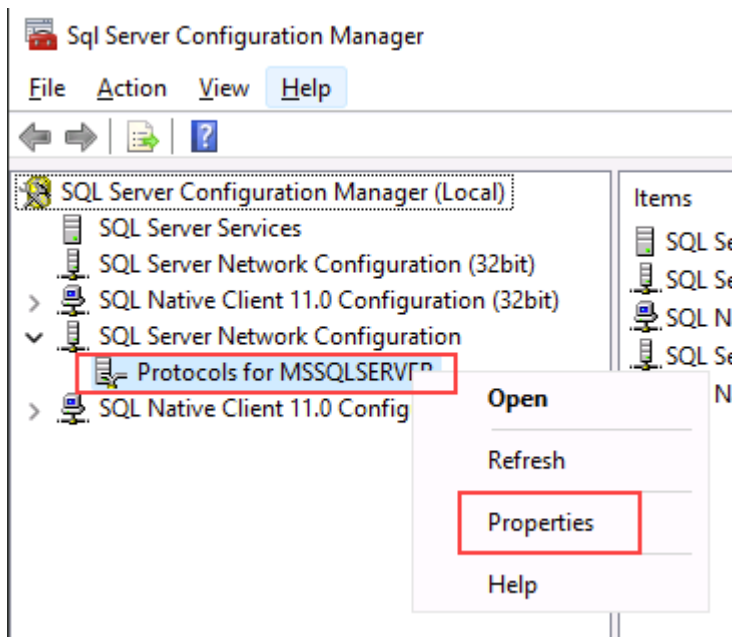
If the correct access isn't provided, restarting SQL Server service fails.

For a complete list of requirements when installing a TLS certificate, see [Enable encrypted connections to the Database Engine](#).

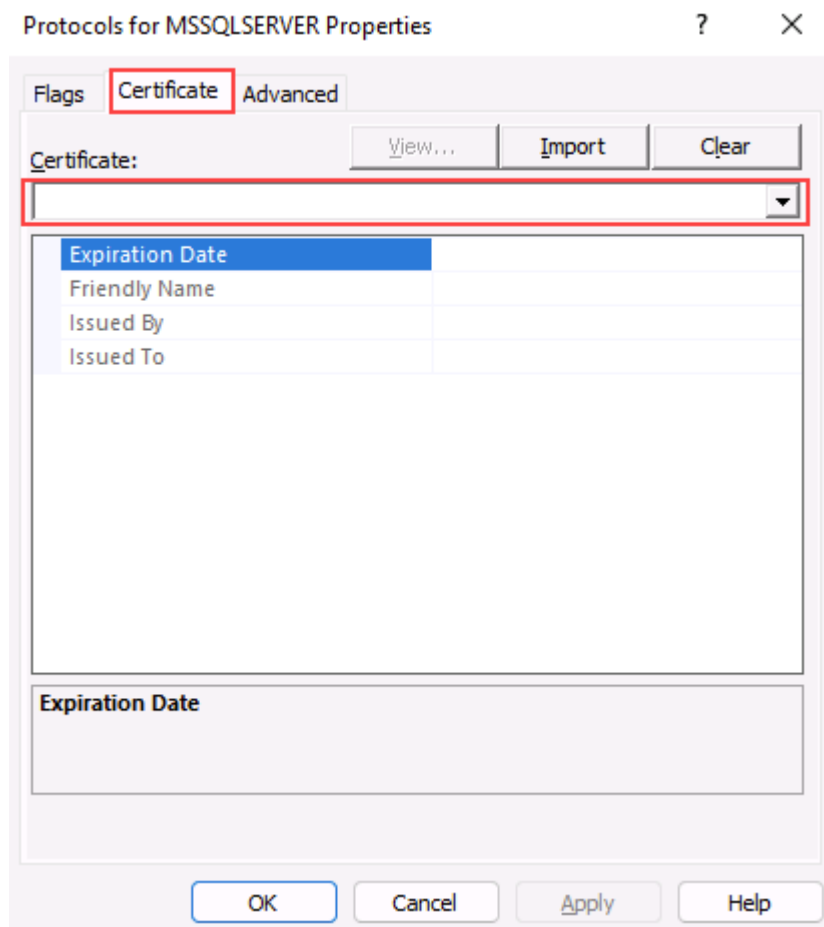
Configure SQL Server instance

You can configure a SQL Server instance to use encrypted connections by following these steps:

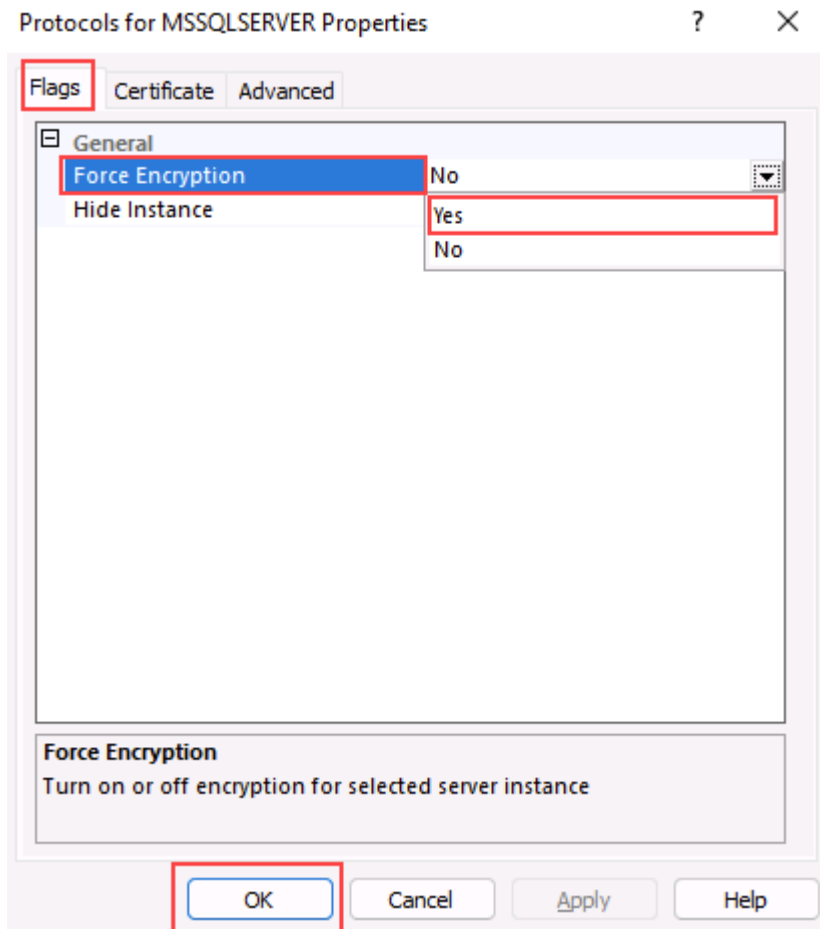
1. On the **SQL Server Configuration Manager**, expand **SQL Server Network Configuration**, right-click **Protocols for <server instance>**, and then select **Properties**.



2. From the **Protocols for <server instance> Properties** dialog box, select the **Certificate** tab, then select the certificate from the **Certificate** drop-down.



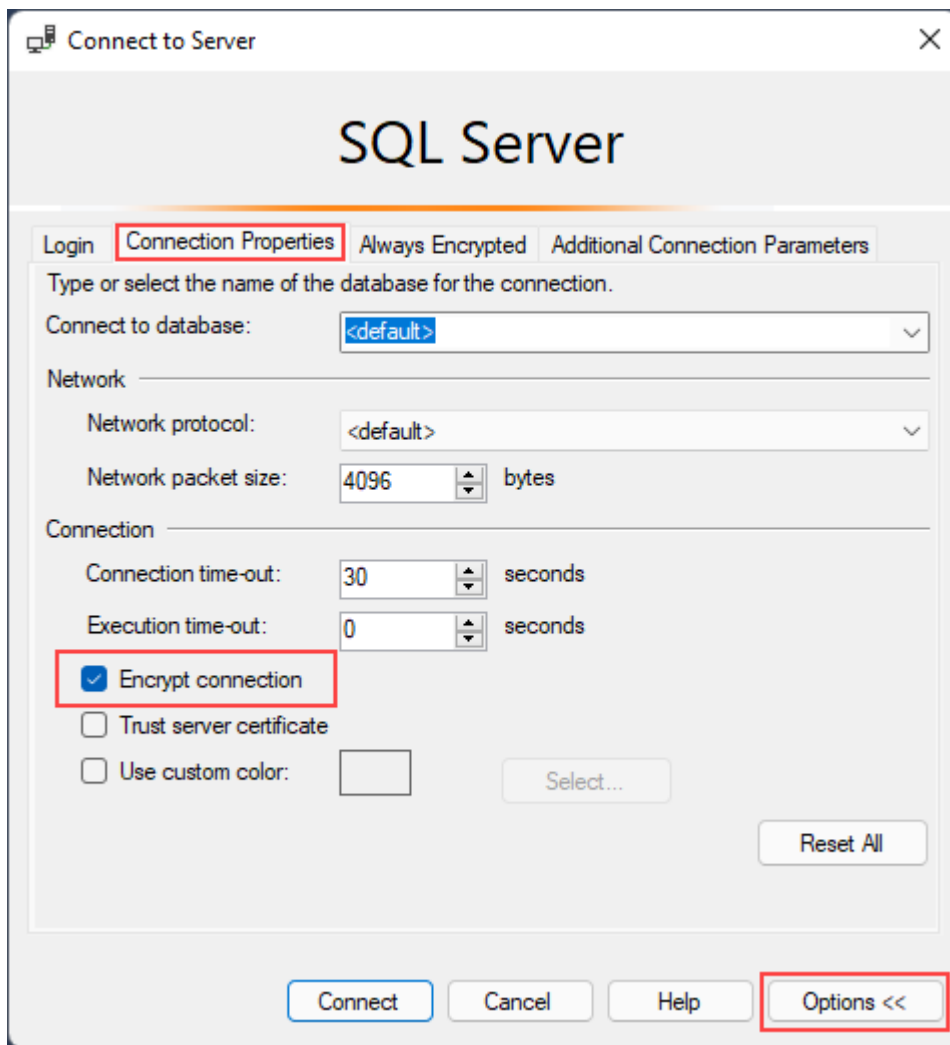
3. On the **Flags** tab, in the **ForceEncryption** property, select **Yes**, and then select **OK**.



4. Restart the SQL Server service.

Once the necessary configuration is in place, you can test the connection through SQL Server Management Studio:

1. In the **Connect to Server** dialog box, complete the connection information, and then select **Options**.
2. On the **Connection Properties** tab, select **Encrypt connection**, and then **Connect**.



All steps must have been executed correctly for you to be able to authenticate through SQL Server Management Studio using TLS.

6. Describe SQL injection

<https://learn.microsoft.com/en-us/training/modules/protect-data-transit-rest/6-describe-sql-injection>

Describe SQL injection

Completed

SQL injection is a common method used for data breaches. The attack involves appending an SQL command to a form field in a web or application front end, usually through a website, with the intent of breaking the original SQL script and executing the injected SQL script. This often occurs when dynamically generated SQL is used within the client application. The primary cause of SQL injection

attacks is poor coding practices in both the client application and database stored procedures. While many developers adopt better practices, SQL injection remains a significant problem due to the prevalence of legacy applications and newer applications built by developers who didn't prioritize SQL injection prevention.

As an example, assume that a front-end web application creates a dynamic SQL statement as follows:

```
SELECT * FROM Orders WHERE OrderId=25
```

This T-SQL is created when the user goes to the sales order history portion of the company's website and enters 25 into the form field for the order ID number. However, suppose the user enters more than just an ID number, for example "25; DELETE FROM Orders;"

In that case, the query sent to your database would be as follows:

```
SELECT * FROM Orders WHERE OrderID=25; DELETE FROM Orders;
```

The way the query in the above example works is that the SQL database is told via the semicolon ";" that the statement has ended and that there's another statement that should be run. The database then processes the next statement as instructed, which would result in the deletion of all rows from the Orders table.

The initial `SELECT` query is run as normal without any errors being generated. However, when you look at the Orders table, you don't see any rows. The second query in the batch, which deletes all the rows, was also executed.

One technique used to prevent SQL injection attacks is to inspect the text of the parameters, or values entered into the form fields, looking for various keywords. However, this solution only provides minimum protection as there are many, many ways to force these attacks to work. Some of these injection techniques include passing in binary data, having the database engine convert the binary data back to a text string, and then executing the string.

You can run the following T-SQL code to see an example of this scenario.

```
DECLARE @v VARCHAR(255)

SELECT @v = cast(0x73705F68656C7064462 AS VARCHAR(255))

EXEC (@v)
```

When accepting data from a user, whether a customer or an employee, it's crucial to validate that the input matches the expected data type to prevent SQL injection attacks. If a number is expected, the client application should verify that the input is indeed a number. If a text string is expected, ensure it is of the correct length and doesn't contain any binary data. The client application should validate all user input. Validation can be done by either informing the user of the issue and allowing them to correct it, or by gracefully handling the error to ensure no commands are sent to the database or file system.

While fixing your application code should always be the priority, there may be cases where it's not possible. In such cases, Advanced Threat Protection can offer an extra layer of security for your sensitive data.

7. Understand Azure Key Vault

<https://learn.microsoft.com/en-us/training/modules/protect-data-transit-rest/7-understand-azure-key-vault>

Understand Azure Key Vault

Completed

Azure Key Vault is a tool used for storing and accessing secrets. Whether they be passwords, certificates or keys, Key Vault acts as a secure area for those secrets to be accessed in a secure fashion, typically programmatically. Key Vault data has its own RBAC policies, separate from the Azure subscription. This means someone who is in the role of subscription admin won't have access to the Key Vault unless explicitly granted.

SQL Server, either within an Azure Virtual Machine or on-premises, supports using Azure Key Vault to store certificates for features such as Transparent Data Encryption, Backup Encryption, or Always Encrypted. While this configuration is complex in an on-premises environment, it's easily managed when using SQL Server on Azure Virtual Machine, as shown in the following image.

Microsoft Azure

Home > New > Marketplace > Get Started > SQL Server 2017 on Windows Server 2016 > Create a virtual machine

Create a virtual machine

Basics Disks Networking Management Advanced **SQL Server settings** Tags Review + create

SECURITY & NETWORKING

* SQL connectivity Private (within Virtual Network) ▼

* Port 1433

SQL AUTHENTICATION

SQL Authentication **Disable** **Enable**

Azure Key Vault integration **Disable** **Enable**

* Key Vault URL **https://contosokeyvault.azure.net** ✓

* Principal name ✓

* Principal secret ✓

* Credential name ✓

In order to configure the Azure Key Vault integration, you need to set the Key Vault URL, the Principal name, the Principal secret, and the name of the credential. This task can be done at the virtual machine creation or to an existing VM.

Configuring SQL Server to connect to Azure Key Vault first requires creating a normal SQL Server login within the instance. Next a Credential needs to be created and mapped to the login. For the identity of the credential, the name of the key vault should be used. For the secret of the credential, use the application ID from Azure Key Vault.

Once the credential is created, an asymmetric key can be created within the Azure Key Vault. An asymmetric key can then be created within the SQL Server database. The key in database can be mapped to the Azure Key Vault asymmetric key using the `CREATE ASYMMETRIC KEY` command with the `FROM PROVIDER` syntax. Once an asymmetric key is created within the database, that key can be used for TDE, or Backup Encryption or Always Encrypted.

8. Exercise: Configure a server-based firewall rule using the Azure portal

<https://learn.microsoft.com/en-us/training/modules/protect-data-transit-rest/8-exercise-configure-azure-sql-database-firewall>

Exercise: Configure a server-based firewall rule using the Azure portal

Completed

In this exercise, you'll learn how to configure a server-based firewall rule using the Azure portal.

Note

To complete this exercise, you'll need an [Azure subscription](#).

Launch the exercise and follow the instructions.

[Launch Exercise](#)

9. Module assessment

<https://learn.microsoft.com/en-us/training/modules/protect-data-transit-rest/9-knowledge-check>

Module assessment

Completed

10. Summary

<https://learn.microsoft.com/en-us/training/modules/protect-data-transit-rest/10-summary>

Summary

Completed

This module explored the encryption options available within Microsoft SQL Server and Azure SQL. Each of the various platforms supports different database encryption options. You also explored these data encryption options and how to configure them.

Now that you've reviewed this module, you should be able to:

- Understand the difference between server and database firewall rules in Azure SQL Database
- Understand how Always Encrypted is used to protect data in transit
- Understand the role of Azure Key Vault in Transparent Data Encryption
- Explain how to enable encrypted connections