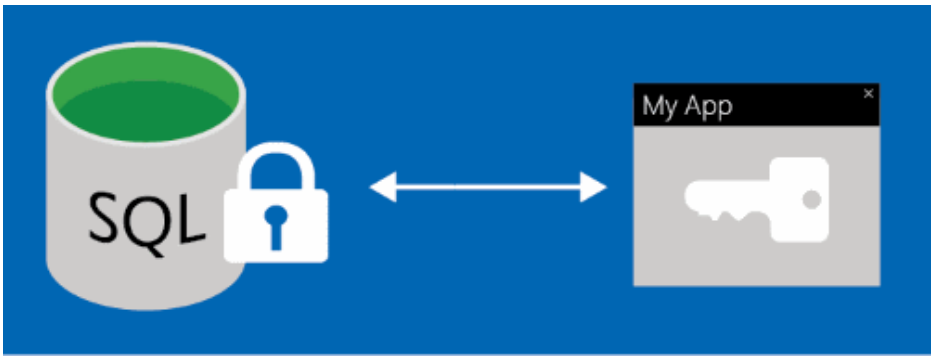
Always Encrypted (Database Engine)

04/24/2017 • 13 minutes to read • Contributors

In this article

Typical Scenarios
How it Works
Selecting Deterministic or Randomized Encryption
Configuring Always Encrypted
Getting Started with Always Encrypted
Feature Details
Database Permissions
Example
See Also

THIS TOPIC APPLIES TO: SQL Server Azure SQL Database Azure SQL Data Warehouse Parallel Data Warehouse



Always Encrypted is a feature designed to protect sensitive data, such as credit card numbers or national identification numbers (for example, U.S. social security numbers), stored in Azure SQL Database or SQL Server databases. Always Encrypted allows clients to encrypt sensitive data inside client applications and never reveal the encryption keys to the Database Engine (SQL Database or SQL Server). 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 administra cloud database operators, or other high-privileged, but unauthorized users, cannot access the encrypted

FEEDBACK

data, Always Encrypted enables customers to confidently store sensitive data outside of their direct control. This allows organizations to encrypt data at rest and in use for storage in Azure, to enable delegation of on-premises database administration to third parties, or to reduce security clearance requirements for their own DBA staff.

Always Encrypted makes encryption transparent to applications. An Always Encrypted-enabled driver installed on the client computer achieves this by automatically encrypting and decrypting sensitive data in the client application. The driver encrypts the data in sensitive columns before passing the data to the Database Engine, and automatically rewrites queries so that the semantics to the application are preserved. Similarly, the driver transparently decrypts data, stored in encrypted database columns, contained in query results.

Always Encrypted is available in SQL Server 2016 and SQL Database. (Prior to SQL Server 2016 SP1, Always Encrypted was limited to the Enterprise Edition.) For a Channel 9 presentation that includes Always Encrypted, see Keeping Sensitive Data Secure with Always Encrypted.

Typical Scenarios

Client and Data On-Premises

A customer has a client application and SQL Server both running on-premises, at their business location. The customer wants to hire an external vendor to administer SQL Server. In order to protect sensitive data stored in SQL Server, the customer uses Always Encrypted to ensure the separation of duties between database administrators and application administrators. The customer stores plaintext values of Always Encrypted keys in a trusted key store which the client application can access. SQL Server administrators have no access to the keys and, therefore, are unable to decrypt sensitive data stored in SQL Server.

Client On-Premises with Data in Azure

A customer has an on-premises client application at their business location. The application operates on sensitive data stored in a database hosted in Azure (SQL Database or SQL Server running in a virtual machine on Microsoft Azure). The customer uses Always Encrypted and stores Always Encrypted keys in a trusted key store hosted on-premises, to ensure Microsoft cloud administrators have no access to sensitive data.

Client and Data in Azure

A customer has a client application, hosted in Microsoft Azure (for example, in a worker role or a web role), which operates on sensitive data stored in a database hosted in Azure (SQL Database or SQL Server running in a virtual machine on Microsoft Azure). Although Always Encrypted does not provide complete isolation of data from cloud administrators, as both the data and keys are exposed to cloud

administrators of the platform hosting the client tier, the customer still benefits from reducing the security attack surface area (the data is always encrypted in the database).

How it Works

You can configure Always Encrypted for individual database columns containing your sensitive data. When setting up encryption for a column, you specify the information about the encryption algorithm and cryptographic keys used to protect the data in the column. Always Encrypted uses two types of keys: column encryption keys and column master keys. A column encryption key is used to encrypt data in an encrypted column. A column master key is a key-protecting key that encrypts one or more column encryption keys.

The Database Engine stores encryption configuration for each column in database metadata. Note, however, the Database Engine never stores or uses the keys of either type in plaintext. It only stores encrypted values of column encryption keys and the information about the location of column master keys, which are stored in external trusted key stores, such as Azure Key Vault, Windows Certificate Store on a client machine, or a hardware security module.

To access data stored in an encrypted column in plaintext, an application must use an Always Encrypted enabled client driver. When an application issues a parameterized query, the driver transparently collaborates with the Database Engine to determine which parameters target encrypted columns and, thus, should be encrypted. For each parameter that needs to be encrypted, the driver obtains the information about the encryption algorithm and the encrypted value of the column encryption key for the column, the parameter targets, as well as the location of its corresponding column master key.

Next, the driver contacts the key store, containing the column master key, in order to decrypt the encrypted column encryption key value and then, it uses the plaintext column encryption key to encrypt the parameter. The resultant plaintext column encryption key is cached to reduce the number of round trips to the key store on subsequent uses of the same column encryption key. The driver substitutes the plaintext values of the parameters targeting encrypted columns with their encrypted values, and it sends the query to the server for processing.

The server computes the result set, and for any encrypted columns included in the result set, the driver attaches the encryption metadata for the column, including the information about the encryption algorithm and the corresponding keys. The driver first tries to find the plaintext column encryption key in the local cache, and only makes a round to the column master key, if it cannot find the key in the cache. Next, the driver decrypts the results and returns plaintext values to the application.

A client driver interacts with a key store, containing a column master key, using a column master key store provider, which is a client-side software component that encapsulates a key store containing the column master key. Providers for common types of key stores are available in client side driver libraries from Microsoft or as standalone downloads. You can also implement your own provider. Always

Encrypted capabilities, including built-in column master key store providers, vary by a driver library and its version.

For details of how to develop applications using Always Encrypted with particular client drivers, see Always Encrypted (client development).

Selecting Deterministic or Randomized Encryption

The Database Engine never operates on plaintext data stored in encrypted columns, but it still supports some queries on encrypted data, depending on the encryption type for the column. Always Encrypted supports two types of encryption: randomized encryption and deterministic encryption.

- Deterministic encryption always generates the same encrypted value for any given plain text value. Using deterministic encryption allows point lookups, equality joins, grouping and indexing on encrypted columns. However, but may also allow unauthorized users to guess information about encrypted values by examining patterns in the encrypted column, especially if there is a small set of possible encrypted values, such as True/False, or North/South/East/West region. Deterministic encryption must use a column collation with a binary2 sort order for character columns.

- Randomized encryption uses a method that encrypts data in a less predictable manner. Randomized encryption is more secure, but prevents searching, grouping, indexing, and joining on encrypted columns.

Use deterministic encryption for columns that will be used as search or grouping parameters, for example a government ID number. Use randomized encryption, for data such as confidential investigation comments, which are not grouped with other records and are not used to join tables. For details on Always Encrypted cryptographic algorithms, see Always Encrypted Cryptography.

Configuring Always Encrypted

The initial setup of Always Encrypted in a database involves generating Always Encrypted keys, creating key metadata, configuring encryption properties of selected database columns, and/or encrypting data that may already exist in columns that need to be encrypted. Please note that some of these tasks are not supported in Transact-SQL and require the use of client-side tools. As Always Encrypted keys and protected sensitive data are never revealed in plaintext to the server, the Database Engine cannot be involved in key provisioning and perform data encryption or decryption operations. You can use SQL Server Management Studio or PowerShell to accomplish such tasks.

Task	SSMS	PowerShell	T-SQL
Provisioning column master keys, column encryption keys and	Yes	Yes	No

encrypted column encryption keys with their corresponding column master keys.			
Creating key metadata in the database.	Yes	Yes	Yes
Creating new tables with encrypted columns	Yes	Yes	Yes
Encrypting existing data in selected database columns	Yes	Yes	No

> **Note**
>
> Make sure you run key provisioning or data encryption tools in a secure environment, on a computer that is different from the computer hosting your database. Otherwise, sensitive data or the keys could leak to the server environment, which would reduce the benefits of the using Always Encrypted.

For details on configuring Always Encrypted see:

- Configure Always Encrypted using SSMS
- Configure Always Encrypted using PowerShell

Getting Started with Always Encrypted

Use the Always Encrypted Wizard to quickly start using Always Encrypted. The wizard will provision the required keys and configure encryption for selected columns. If the columns, you are setting encryption for, already contain some data, the wizard will encrypt the data. The following example demonstrates the process for encrypting a column.

> **Note**
>
> For a video that includes using the wizard, see Getting Started with Always Encrypted with SSMS.

1. Connect to an existing database that contains tables with columns you wish to encrypt using the **Object Explorer** of Management Studio, or create a new database, create one or more tables with columns to encrypt, and connect to it.
2. Right-click your database, point to **Tasks**, and then click** Encrypt Columns** to open the **Always Encrypted Wizard**.

3. Review the **Introduction** page, and then click **Next**.
4. On the **Column Selection** page, expand the tables, and select the columns that you want to encrypt.
5. For each column selected for encryption, set the **Encryption Type** to either *Deterministic* or *Randomized*.
6. For each column selected for encryption, select an **Encryption Key**. If you have not previously created any encryption keys for this database, select the default choice of a new auto-generated key, and then click **Next**.
7. On the **Master Key Configuration** page, select a location to store the new key, and select a master key source, and then click **Next**.
8. On the **Validation** page, choose whether to run the script immediately or create a PowerShell script, and then click **Next**.
9. On the **Summary** page, review the options you have selected, and then click **Finish**. Close the wizard when completed.

Feature Details

- Queries can perform equality comparison on columns encrypted using deterministic encryption, but no other operations (for example, greater/less than, pattern matching using the LIKE operator, or arithmetical operations).

- Queries on columns encrypted by using randomized encryption cannot perform operations on any of those columns. Indexing columns encrypted using randomized encryption is not supported.

- A column encryption key can have up to two different encrypted values, each encrypted with a different column master key. This facilitates column master key rotation.

- Deterministic encryption requires a column to have one of the *binary2* collations.

- After changing the definition of an encrypted object, execute sp_refresh_parameter_encryption to update the Always Encrypted metadata for the object.

Always Encrypted is not supported for the columns with the below characteristics (for example, the *Encrypted WITH* clause cannot be used in **CREATE TABLE/ALTER TABLE** for a column, if any of the following conditions apply to the column):

- Columns using one of the following datatypes: **xml**, **timestamp/rowversion**, **image**, **ntext**, **text**, **sql_variant**, **hierarchyid**, **geography**, **geometry**, alias, user defined-types.
- FILESTREAM columns
- Columns with the IDENTITY property
- Columns with ROWGUIDCOL property
- String (varchar, char, etc.) columns with non-bin2 collations

- Columns that are keys for nonclustered indices using a randomized encrypted column as a key column (deterministic encrypted columns are fine)
- Columns that are keys for clustered indices using a randomized encrypted column as a key column (deterministic encrypted columns are fine)
- Columns that are keys for fulltext indices containing encrypted columns both randomized and deterministic
- Columns referenced by computed columns (when the expression does unsupported operations for Always Encrypted)
- Sparse column set
- Columns that are referenced by statistics
- Columns using alias type
- Partitioning columns
- Columns with default constraints
- Columns referenced by unique constraints when using randomized encryption (deterministic encryption is supported)
- Primary key columns when using randomized encryption (deterministic encryption is supported)
- Referencing columns in foreign key constraints when using randomized encryption or when using deterministic encryption, if the referenced and referencing columns use different keys or algorithms
- Columns referenced by check constraints
- Columns in tables that use change data capture
- Primary key columns on tables that have change tracking
- Columns that are masked (using Dynamic Data Masking)
- Columns in Stretch Database tables. (Tables with columns encrypted with Always Encrypted can be enabled for Stretch.)
- Columns in external (PolyBase) tables (note: using external tables and tables with encrypted columns in the same query is supported)
- Table-valued parameters targeting encrypted columns are not supported.

The following clauses cannot be used for encrypted columns:

- FOR XML
- FOR JSON PATH

The following features do not work on encrypted columns:

- Transactional or merge replication
- Distributed queries (linked servers)

Tool Requirements

- SQL Server Management Studio can decrypt the results retrieved from encrypted columns if you connect with the *column encryption setting=enabled* in the **Additional Properties** tab of the **Connect to Server** dialog. Requires at least SQL Server Management Studio version 17 to insert, update, or filter encrypted columns.

- Encrypted connections from `sqlcmd` require at least version 13.1, which is available from the Download Center.

Database Permissions

There are four permissions for Always Encrypted:

- `ALTER ANY COLUMN MASTER KEY` (Required to create and delete a column master key.)
- `ALTER ANY COLUMN ENCRYPTION KEY` (Required to create and delete a column encryption key.)
- `VIEW ANY COLUMN MASTER KEY DEFINITION` (Required to access and read the metadata of the column master keys to manage keys or query encrypted columns.)
- `VIEW ANY COLUMN ENCRYPTION KEY DEFINITION` (Required to access and read the metadata of the column encryption key to manage keys or query encrypted columns.)

  The following table summarizes the permissions required for common actions.

Scenario	`ALTER ANY COLUMN MASTER KEY`	`ALTER ANY COLUMN ENCRYPTION KEY`	`VIEW ANY COLUMN MASTER KEY DEFINITION`	`VIEW ANY COLUMN ENCRYPTION KEY DEFINITION`
Key management (creating/changing /reviewing key metadata in the database)	X	X	X	X
Querying encrypted columns			X	X

Important notes:

- The permissions apply to actions using Transact-SQL, Management Studio (dialog boxes and wizard), or PowerShell.
- The two *VIEW* permissions are required when selecting encrypted columns, even if the user does not have permission to decrypt the columns.
- In SQL Server, both *VIEW* permissions are granted by default to the `public` fixed database role. A database administrator may choose to revoke (or deny) the *VIEW* permissions to the `public` role and grant them to specific roles or users to implement more restricted control.

- In SQL Database, the *VIEW* permissions are not granted by default to the `public` fixed database role. This enables certain existing, legacy tools (using older versions of DacFx) to work properly. Consequently, to work with encrypted columns (even if not decrypting them) a database administrator must explicitly grant the two *VIEW* permissions.

Example

The following Transact-SQL creates column master key metadata, column encryption key metadata, and a table with encrypted columns. For information how to create the keys, referenced in the metadata, see:

- Configure Always Encrypted using SSMS
- Configure Always Encrypted using PowerShell

Copy

```
CREATE COLUMN MASTER KEY MyCMK
WITH (
     KEY_STORE_PROVIDER_NAME = 'MSSQL_CERTIFICATE_STORE',
     KEY_PATH = 'Current User/Personal/f2260f28d909d21c642a3d8e0b45a830e79a1420'
   );
----------------------------------------------
CREATE COLUMN ENCRYPTION KEY MyCEK
WITH VALUES
(
    COLUMN_MASTER_KEY = MyCMK,
    ALGORITHM = 'RSA_OAEP',
    ENCRYPTED_VALUE = 0x01700000016C006F00630061006C006D006100630068006900 6E0065002F006D00
);
----------------------------------------------
CREATE TABLE Customers (
    CustName nvarchar(60)
        COLLATE  Latin1_General_BIN2 ENCRYPTED WITH (COLUMN_ENCRYPTION_KEY = MyCEK,
        ENCRYPTION_TYPE = RANDOMIZED,
        ALGORITHM = 'AEAD_AES_256_CBC_HMAC_SHA_256'),
    SSN varchar(11)
        COLLATE  Latin1_General_BIN2 ENCRYPTED WITH (COLUMN_ENCRYPTION_KEY = MyCEK,
        ENCRYPTION_TYPE = DETERMINISTIC ,
        ALGORITHM = 'AEAD_AES_256_CBC_HMAC_SHA_256'),
    Age int NULL
);
GO
```

See Also

CREATE COLUMN MASTER KEY (Transact-SQL)
CREATE COLUMN ENCRYPTION KEY (Transact-SQL)
CREATE TABLE (Transact-SQL)

column_definition (Transact-SQL)
sys.column_encryption_keys (Transact-SQL)
sys.column_encryption_key_values (Transact-SQL)
sys.column_master_keys (Transact-SQL)
sys.columns (Transact-SQL)
Always Encrypted Wizard
Migrate Sensitive Data Protected by Always Encrypted
Always Encrypted (client development)
Always Encrypted Cryptography
Configure Always Encrypted using SSMS Configure Always Encrypted using PowerShell
sp_refresh_parameter_encryption (Transact-SQL)

> **Note**
>
> The feedback system for this content will be changing soon. Old comments will not be carried over. If content within a comment thread is important to you, please save a copy. For more information on the upcoming change, we invite you to read our blog post.

6 Comments

Sign in **14 people listening**

+ Follow Post comment as...

Conrad_B_Seelye Dec 4, 2017

How is it that you can execute a TSQL statement for "CREATE COLUMN ENCRYPTION KEY MyCEK ", but the table above states that TSQL cannot be used for "Provisioning column master keys, column encryption keys and encrypted column encryption keys with their corresponding column master keys."

Like Reply

Srimjam Jul 5, 2017

Always Encrypted is introduced to get the data encrypted for even DBA's who can be potential reason for data-leaks.

Though data is encrypted, DBA will use column encryption setting=enabled and can view all data. How is this taken care , which is missing on whole documentation.

Like Reply

MarekFes Jul 13, 2017