Microsoft Database Encryption

I compiled the information below during my research into Always Encrypted as I found it useful in my understanding and testing.

To summarize what I have been able to test and verify at this time:

Using the wizard, it is very easy to encrypt columns that meet the criteria for Always Encrypted.

Once encrypted, SSMS cannot be used to insert data into encrypted columns because it does not use ADO.NET 4.6, JDBC 6.0 and ODBC 13.1, the only drivers that support Always Encrypted at this time.

Once a database has encrypted columns, it can be backed up and moved to another server and the columns will still be encrypted.

Unless the certificates are exported to other systems, the original system/user that has the certificate is the only system that can remove encryption – even if the database was moved to a different server.

Parameterized SQL statements must be used to insert or update data in encrypted columns.

Columns can have encryption removed easily by the controlling system.

Once the encryption is removed, the data can be updated normally.

To view Encrypted data in SSMS, the certificate must be installed on the system running SSMS and a connection string must be added in the advanced tab of SSMS: column encryption setting=enabled.

There are specific requirements that need to be followed to use SSIS with Always Encrypted columns, mainly you must connect using the ADO.NET driver and the connection string must reference the certificate (details are given in the document).

Two types of encryption are used – random and deterministic. Deterministic must be used on columns that are used for joins and searches. Deterministic will change the values but not randomly i.e. ac may change to dfe but it will be like that in every column ac has appeared.

# Current – Transparent Data Encryption (TDE)

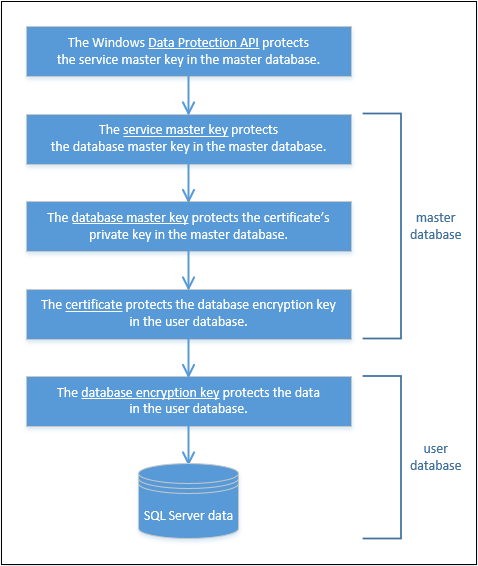
What it does

Transparent Data Encryption (TDE) is a form of encryption that is native to Microsoft SQL Server.

Microsoft offers **TDE** as part of its Microsoft **SQL** Server 2008, 2008 R2, 2012, 2014 and 2016. **TDE** is only **supported** on the Evaluation, Developer, Enterprise and Datacenter **editions** of Microsoft **SQL**Server. (From Wikipedia)

TDE protects data at rest. What this means is that the entire database and its backups are secured by a data-encryption algorithm. Any attempt to read or use the file without the encryption keys such as restoring or attaching the database would result in fatal errors.

The graphic below, taken from an article, illustrates this.



From Robert Sheldon - <https://www.red-gate.com/simple-talk/sql/sql-development/encrypting-sql-server-transparent-data-encryption-tde/>

What TDE does not do

As good as TDE is as a security measure, it can be defeated by a simple SQL statement run by an authenticated user – Select Into YourTableName from TDEEncryptedTable. The reason for this has to do with the Transparent part of TDE – the encryption is transparently removed as the data is queried.

This means that users who have access to the database can query the encrypted database and still see all the data. The same would be true for hackers who managed to break in as authenticated users. Sensitive data like social security numbers could be exposed to casual users who really would not need to see the data as part of their jobs.

There are other security measures that can be used to protect this data but these solutions are more tactical in nature and not strategic like TDE.

Always Encrypted

Quoted from Microsoft:

*Always Encrypted is a feature designed to protect sensitive data, such as credit card numbers or national identification numbers (e.g. 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).*

Always Encrypted protects sensitive data by storing the encryption keys locally with end users. 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.)

At the time of writing Always Encrypted is only supported with ADO.NET 4.6, JDBC 6.0 and ODBC 13.1 but expect other drivers to become available.  
  
The calling application (including SSMS) must also have an extra parameter in the connection string: column encryption setting=enabled.  
  
To edit the data you must use a parameterized query, you cannot update it using a “normal” update query. Additionally you can’t update encrypted tables using SSMS, it must be done using one of the supported drivers such as PowerShell or a .NET application using ADO.NET 4.6.

How it works

Again from Microsoft:

*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.*

Applications must be written to use a client diver that interacts with the local or master key store to have access the encrypted columns via SQL statements.

From Microsoft:

*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.*

The following chart from Microsoft shows what applications to use to deploy Always Encrypted:

| Task | SSMS | PowerShell | T-SQL |
| --- | --- | --- | --- |
| Provisioning column master keys, column encryption keys and encrypted column encryption keys with their corresponding column master keys. | Yes | Yes | No |
| 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 |

It is recommended to provision Always Encrypted from a server in a secure environment or that is apart from the server hosting the databases to prevent keys or sensitive data leaking to the server environment.

# Always Encrypted Limitations

From Microsoft:

*Feature Details*

* *Queries can perform equality comparison on columns encrypted using deterministic encryption, but no other operations (e.g. 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*](https://docs.microsoft.com/en-us/sql/relational-databases/collations/collation-and-unicode-support)*.*
* *After changing the definition of an encrypted object, execute [sp\_refresh\_parameter\_encryption](https://docs.microsoft.com/en-us/sql/relational-databases/system-stored-procedures/sp-refresh-parameter-encryption-transact-sql) to update the Always Encrypted metadata for the object.*

*Always Encrypted is not supported for the columns with the below characteristics (e.g. 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*](http://go.microsoft.com/fwlink/?LinkID=825643)*.*

*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.*

# Migrating Data Protected By Always Encrypted

From Microsoft:

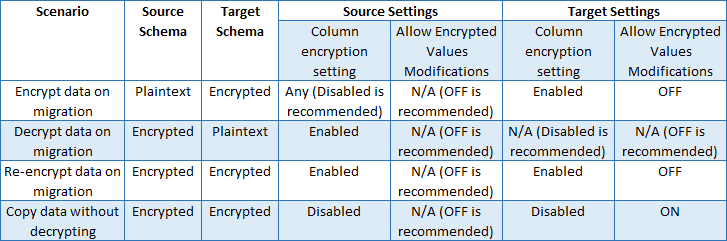
*To load encrypted data without performing metadata checks on the server during bulk copy operations, create the user with the* ***ALLOW\_ENCRYPTED\_VALUE\_MODIFICATIONS*** *option. This option is intended to be used by legacy tools from versions of SQL Server older than SQL Server 2016 (such as bcp.exe) or by using third-party Extract-Transform-Load (ETL) work flows that cannot use Always Encrypted. This allows a user to securely move encrypted data from one set of tables, containing encrypted columns, to another set of tables with encrypted columns (in the same or a different database).*

*The ALLOW\_ENCRYPTED\_VALUE\_MODIFICATIONS Option*

*Both* [*CREATE USER*](https://msdn.microsoft.com/library/ms173463.aspx) *and* [*ALTER USER*](https://msdn.microsoft.com/library/ms176060.aspx) *have an ALLOW\_ENCRYPTED\_VALUE\_MODIFICATIONS option. When set to ON (the default is OFF), this option suppresses cryptographic metadata checks on the server in bulk copy operations, which enables the user to bulk copy encrypted data between tables or databases, without decrypting the data.*

*Data Migration Scenarios*

*The following table shows the recommended settings appropriate for several migration scenarios.*

**

*Bulk Loading of Encrypted Data*

*Use the following process to load encrypted data.*

1. *Set the option to ON for the user in the database that is the target for the bulk copy operation. For example:*

*Copy*

*ALTER USER Bob WITH ALLOW\_ENCRYPTED\_VALUE\_MODIFICATIONS = ON;*

1. *Run your bulk copy application or tool connecting as that user. (If your application uses an Always Encrypted enabled client driver, make sure the connection string for the data source does not contain* ***column encryption setting=enabled*** *to ensure the data retrieved from encrypted columns remains encrypted. For more information, see* [*Always Encrypted (client development)*](https://docs.microsoft.com/en-us/sql/relational-databases/security/encryption/always-encrypted-client-development)*.)*
2. *Set the ALLOW\_ENCRYPTED\_VALUE\_MODIFICATIONS option back to OFF. For example:*

*Copy*

*ALTER USER Bob WITH ALLOW\_ENCRYPTED\_VALUE\_MODIFICATIONS = OFF;*

*Potential for Data Corruption*

*Improper use of this option can lead to data corruption. The* ***ALLOW\_ENCRYPTED\_VALUE\_MODIFICATIONS*** *option allows the user to insert any data into encrypted columns in the database, including data that is encrypted with different keys, incorrectly encrypted, or not encrypted at all. If the user accidently copies the data that is not correctly encrypted using the encryption scheme (column encryption key, algorithm, encryption type) set up for the target column, you will not be able to decrypt the data (the data will be corrupted). This option must be used carefully, as it can lead to corrupting data in the database.*

*The following scenario demonstrates how improperly importing data could lead to data corruption:*

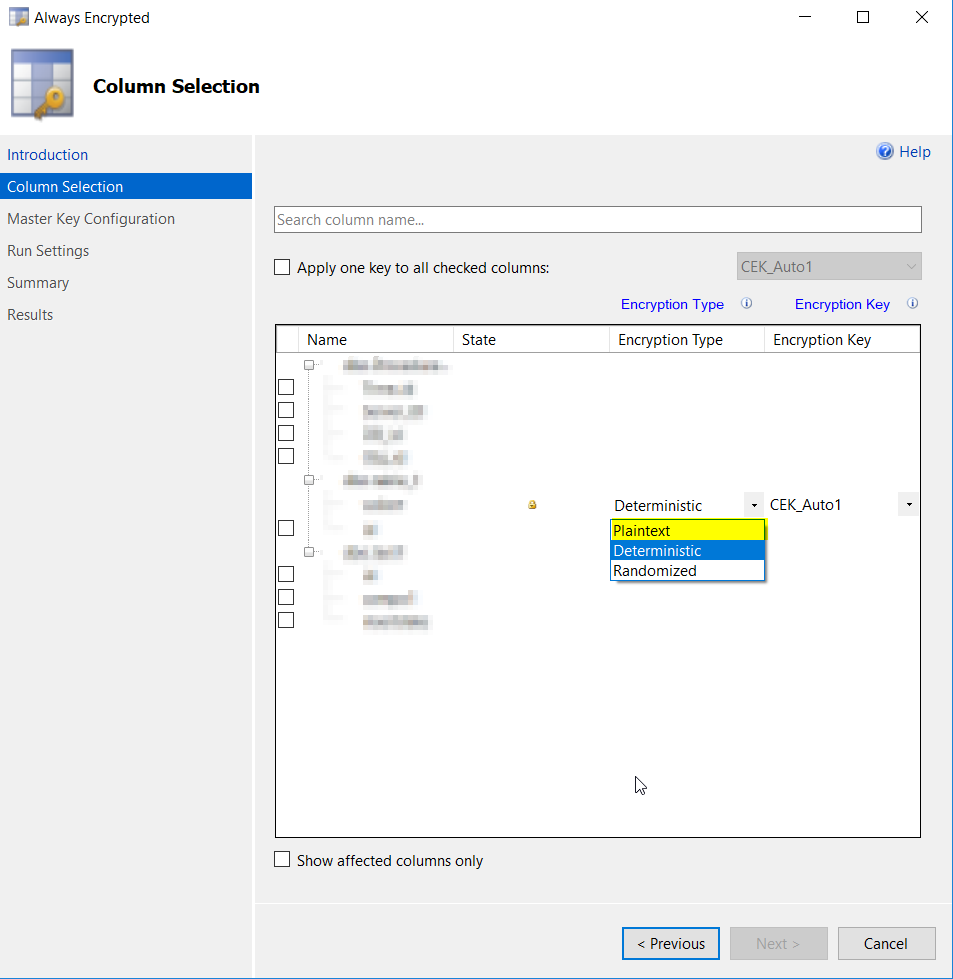
1. *The option is set to ON for a user.*
2. *The user runs the application that connects to the database. The application uses bulk APIs to insert plain text values to encrypted columns. The application expects an Always Encrypted-enabled client driver to encrypt the data on insert. However, the application is misconfigured, so that either it ends up using a driver that does not support Always Encrypted or the connection string does not contain* ***column encryption setting=enabled****.*
3. *The application sends plaintext values to the server. As cryptographic metadata checks are disabled in the server for the user, the server lets the incorrect data (plaintext instead of correctly encrypted ciphertext) to be inserted into an encrypted column.*
4. *The same or another application connects to the database using an Always Encrypted-enabled driver and with* ***column encryption setting=enabled*** *in the connection string, and retrieves the data. The application expects the data to be transparently decrypted. However, the driver fails to decrypt the data because the data is incorrect ciphertext.*

*Best practice*

* *Use designated user accounts for long running workloads using this option.*
* *For short running bulk copy applications or tools that need to move encrypted data without decrypting it, set the option to ON immediately before running the application and set it back to OFF immediately after running the operation.*
* *Do not use this option for developing new applications. Instead, use a client driver (such as ADO 4.6.1) that offers an API for suppressing cryptographic metadata checks for a single session.*

# Removing Always Encrypted From a Column

Run the Encryption Wizard and set the "Encryption Type" to plaintext.



# Implementing Always Encrypted

From Microsoft:

*Use the*[*Always Encrypted Wizard*](https://docs.microsoft.com/en-us/sql/relational-databases/security/encryption/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.*

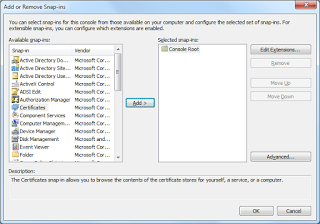
1. *Connect to an existing database that contains tables with columns you wish to encrypt using the****Object Explorer****of Management Studio, or create a new database, create one or more tables with columns to encrypt, and connect to it.*
2. *Right-click your database, point to****Tasks****, and then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

# Managing your Certificate

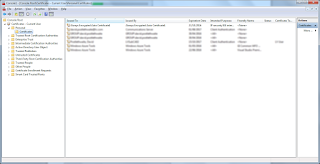
Your certificate is held in your certificate store.

To view your certificate store type “mmc” into the Windows Run box on the Start menu.

In the management console window select File -> Add / Remove Snap-ins and select Certificate from the list and then select current user

[](https://3.bp.blogspot.com/-2PNHSuiuQiE/WPnS6Rh3kzI/AAAAAAAABqk/7tJD1Lbe93o9QwpDeNsE80m2tPWyP-mugCEw/s1600/AE_9.png)

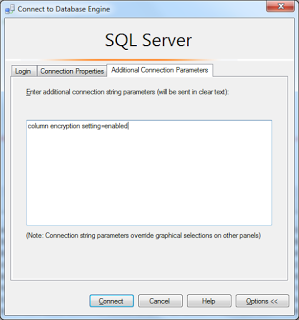
Click OK and you will now see your certificates on your PC.

[](https://1.bp.blogspot.com/-6uAM2lvaqoY/WPnS5LkXtAI/AAAAAAAABqk/h2vqz8ybxmgj2T90X3j9D4U9AzaGhqJ8QCEw/s1600/AE_10.png)

Drill down Personal and you will see the Always Encrypted Certificate. Export this and distribute it to all those who need to view the data. And keep it somewhere safe, without this certificate your data will not be readable, ever.

# Viewing Unencrypted Data

In SSMS reconnect your query but add the parameter “column encryption setting=enabled” to your connection parameters. Run your select statement again and you will now see the data in clear text.

[](https://4.bp.blogspot.com/-LmH8ouum5ho/WPnS4x68GoI/AAAAAAAABqk/jLyYQ0cVFyoeOTbKbN8XVnRsIYQTv1bPACEw/s1600/AE_11.png)

But if you try and insert or update data you will still get an error

insert into Customers VALUES (1,'David','Postlethwaite','12345','1111-1233-1231-1233','david@clunyweb.co.uk','426312',1)

Msg 206, Level 16, State 2, Line 93

Operand type clash: varchar is incompatible with varchar(8000) encrypted with (encryption\_type = 'DETERMINISTIC', encryption\_algorithm\_name = 'AEAD\_AES\_256\_CBC\_HMAC\_SHA\_256', column\_encryption\_key\_name = 'CEK\_Auto1', column\_encryption\_key\_database\_name = 'EncryptedDB') collation\_name = 'Latin1\_General\_CI\_AS'As I said previously, you cannot update encrypted data using a standard query, even in SSMS.

Updating Data using Stored Procedure

To be able to update your table you must use a parameterised query. Below is an example

As I said previously, you cannot update encrypted data using a standard query, even in SSMS.

# Updating Data using Stored Procedure

To be able to update your table you must use a parameterised query. Below is an example

DROP PROCEDURE IF EXISTS dbo.AddCustomer

  GO

  CREATE PROCEDURE dbo.AddCustomer

  @CustomerID int,

  @FirstName nvarchar(25),

  @LastName nvarchar(25),

  @SIN nvarchar(11),

  @CreditCardNumber nvarchar(25),

  @EmailAddress nvarchar(50),

  @PhoneNumber nvarchar(25),

  @TerritoryID int

AS

BEGIN

 INSERT INTO [dbo].[Customers]

           ([CustomerID]

           ,[FirstName]

           ,[LastName]

           ,[SIN]

           ,[CreditCardNumber]

           ,[EmailAddress]

           ,[PhoneNumber]

           ,[TerritoryID])

     VALUES

           (@CustomerID,

           @FirstName,

           @LastName,

           @SIN,

           @CreditCardNumber,

           @EmailAddress,

           @PhoneNumber,

           @TerritoryID)

END

Running the procedure from SSMS will not work. SSMS does not use one of the supported connection types such as ADO to connect to the database.

DECLARE @CustomerID int,

@FirstName nvarchar(25),

@LastName nvarchar(25),

@SIN nvarchar(11),

@CreditCardNumber nvarchar(25),

@EmailAddress nvarchar(50),

@PhoneNumber nvarchar(25),

@TerritoryID int

SET @CustomerID = 1

SET @FirstName = 'David'

SET @LastName = 'Postlethwaite'

SET @SIN = '12345-3-ee-3'

SET @CreditCardNumber = '1111-1233-1231-1233'

SET @EmailAddress = 'david@clunyweb.co.uk'

SET @PhoneNumber = '406555'

SET @TerritoryID = 1

execdbo.AddCustomer @CustomerID,@FirstName,@LastName,@SIN,@CreditCardNumber,@EmailAddress,@PhoneNumber,@TerritoryID

Msg 33299, Level 16, State 6, Line 146

Encryption scheme mismatch for columns/variables '@SIN'. The encryption scheme for the columns/variables is (encryption\_type = 'PLAINTEXT') and the expression near line '0' expects it to be (encryption\_type = 'DETERMINISTIC', encryption\_algorithm\_name = 'AEAD\_AES\_256\_CBC\_HMAC\_SHA\_256', column\_encryption\_key\_name = 'CEK\_Auto1', column\_encryption\_key\_database\_name = 'EncryptedDB') (or weaker).

To be able to insert data you must connect using one of the supported connections currently ADO.NET 4.6, JDBC 6.0 and ODBC 13.1

# Using PowerShell to update Data

Start PowerShell IDE . The following command will read the data from your encrypted table.

$conn= New-ObjectSystem.Data.SqlClient.SqlConnection

$conn.ConnectionString = "Server=sqlserver1;Database=EncryptedDB;Integrated Security=SSPI; Column Encryption Setting=enabled;"

$conn.Open()

$Datatable = New-ObjectSystem.Data.DataTable

$Command= New-ObjectSystem.Data.SQLClient.SQLCommand

$Command.Connection = $conn

$Command.CommandText = "Select \* from customers"

$Reader= $Command.ExecuteReader()

$Datatable.Load($Reader)

$Datatable

$conn.Close()

The following will run the parametrised stored procedure we created earlier. You will notice that the two encrypted values require a full set of parameters values to work correctly, it is not enough just to pass the parameter and the new value like the non encrypted columns.

$SqlConn1= New-ObjectSystem.Data.SqlClient.SqlConnection

$SqlConn1.ConnectionString = "Server=SQLServer1;Database=EncryptedDB;Integrated Security=SSPI; Column Encryption Setting=enabled;"

$SqlConn1.Open()

$SqlCmd1= New-ObjectSystem.Data.SqlClient.SqlCommand

$sqlcmd.CommandType = [System.Data.CommandType]::StoredProcedure

$SqlCmd1.Connection = $SqlConn1

$SqlCmd1.CommandText = "dbo.AddCustomer"

$SqlCmd1.Parameters.Add("@CustomerID",[system.data.SqlDbType]::VarChar) |out-Null

$SqlCmd1.Parameters['@CustomerID'].Direction =[system.data.ParameterDirection]::Input

$SqlCmd1.Parameters['@CustomerID'].value = 2

$SqlCmd1.Parameters.AddWithValue("@FirstName", 'David')

$SqlCmd1.Parameters.AddWithValue("@LastName", 'Postlethwaite')

$SSN= New-Object-TypeName System.Data.SqlClient.SqlParameter

$SSN.ParameterName = "@SSN"

$SSN.SqlDbType = [System.Data.SqlDbType]::VarChar

$SSN.Size=11

$SSN.Direction = [System.Data.ParameterDirection]::Input

$SSN.Value = '12345-3-ee-3'

$SqlCmd1.Parameters.Add($SSN);

$CreditCardNumber = New-Object -TypeName System.Data.SqlClient.SqlParameter

$CreditCardNumber.ParameterName = "@CreditCardNumber"

$CreditCardNumber.SqlDbType = [System.Data.SqlDbType]::VarChar

$CreditCardNumber.Size = 25

$CreditCardNumber.Direction = [System.Data.ParameterDirection]::Input

$CreditCardNumber.Value = '1111-1233-1231-1233'

$SqlCmd1.Parameters.Add($CreditCardNumber);

$SqlCmd1.Parameters.AddWithValue("@EmailAddress", 'ddd@clunyweb.ca')

$SqlCmd1.Parameters.AddWithValue("@PhoneNumber", '0111 421233')

$SqlCmd1.Parameters.AddWithValue("@TerritoryID", 1)

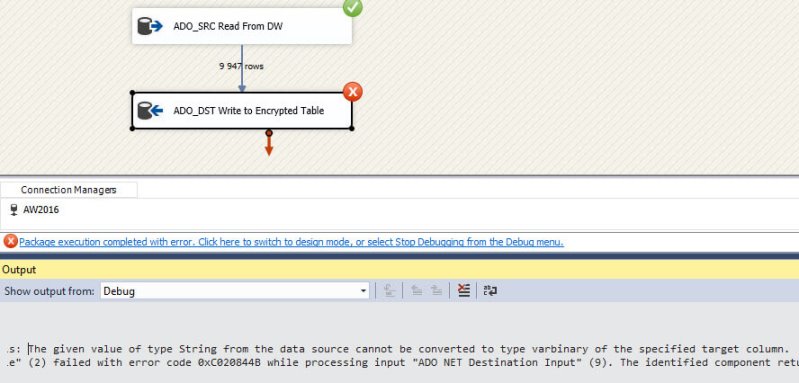
$SqlCmd1.ExecuteNonQuery();

# $SqlConn1.Close()

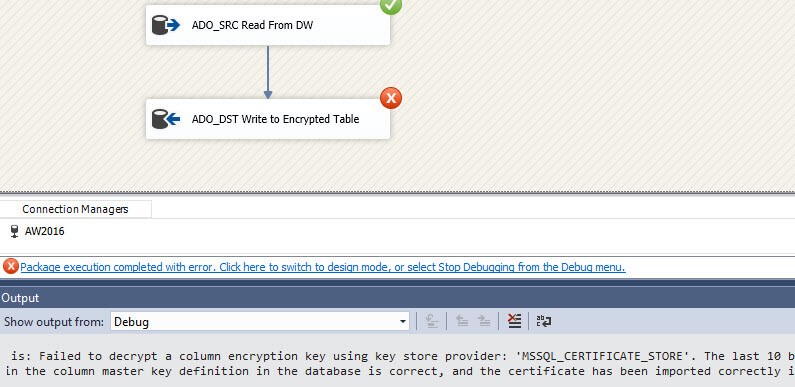
# Writing data to an encrypted table with SSIS

Using a Data Flow

Only ADO.NET connection managers support the additional connection property. When it isn't specified, writing data to an encrypted table will lead to an error:



If you set the connection property Column Encryption Setting to enabled, but haven't installed the certificate, you'll get the following error:



If everything is configured correctly, we can start writing data to the table. We will use the following query to fetch the source data:

SELECT

[CustomerID] = CONVERT(INT,REPLACE([CustomerAlternateKey],'AW',''))

,[FirstName]

,[LastName]

,[SSN] = N'DummySSN'

,[CreditCardNumber] = N'DummyCreditCard'

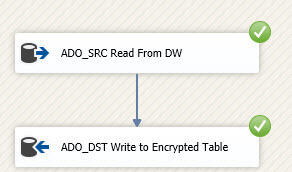
,[EmailAddress]

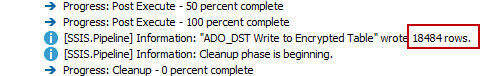
,[PhoneNumber] = [Phone]

,[TerritoryID] = -1

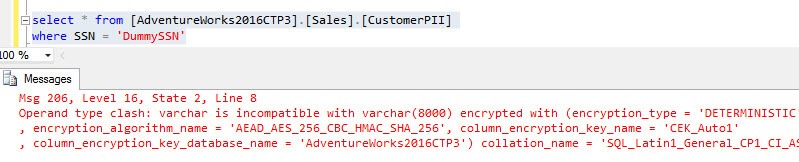
FROM [AdventureworksDW2016CTP3].[dbo].[DimCustomer];

Writing data to the table is now just the same as in any other data flow. You just have to remember to use ADO.NET components instead of OLE DB.

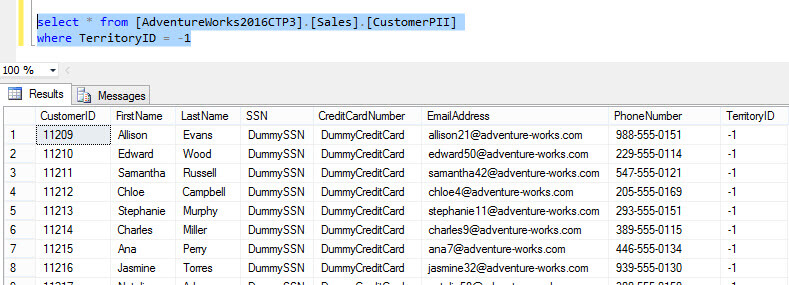




Let's verify if the rows actually made it correctly into the table. You cannot filter on the encrypted columns themselves.



When we filter on another column, we can see the rows made it to the destination table.



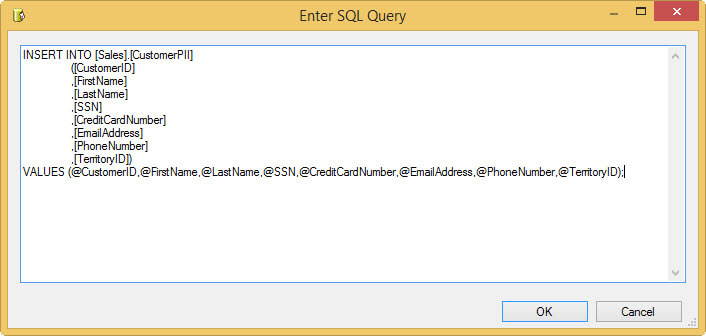
To clean-up the destination table, you can use the following SQL statement:

DELETE FROM [AdventureWorks2016CTP3].[Sales].[CustomerPII]

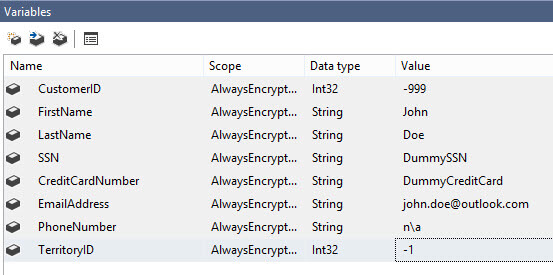
WHERE TerritoryID = -1;

Using an Execute SQL Task

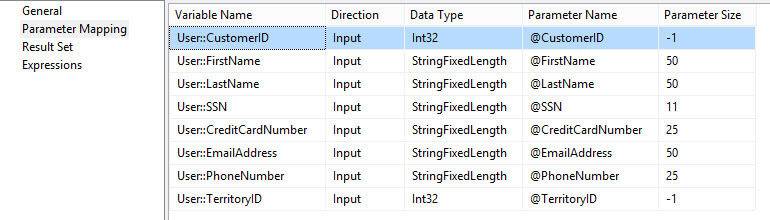
In this section we will write data to the Always Encrypted table using an Execute SQL Task. However, we cannot use a regular SQL statement, as this will fail as demonstrated at the start of this tip. We will need a SQL statement where the values are passed in as parameters. The statement looks like this:



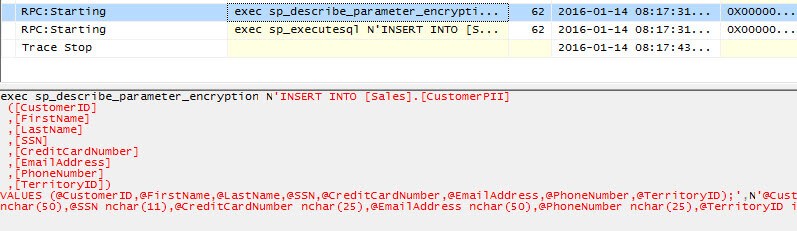
Since we're using an ADO.NET connection, we can use named parameters in the SQL script instead of question marks like in OLE DB. First we need to create variables that will hold the different values.



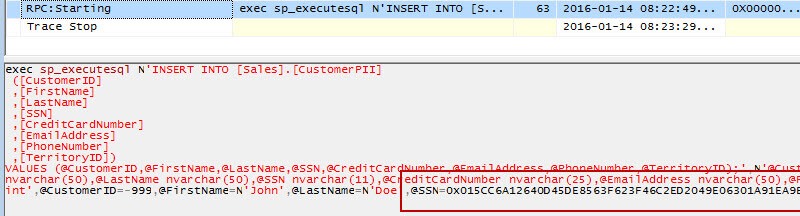
Next, map those variables against the parameters in the parameter mapping pane of the Execute SQL Task:



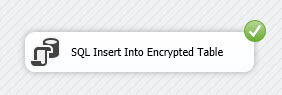
Let's take a look in Profiler to find out what SSIS is sending to SQL Server when the package is running. Two statements are sent to SQL Server. The first one uses an extended system stored procedure to find out if parameters are encrypted.



The second one sends the actual insert statement to SQL Server. Although the encrypted columns are small Unicode text columns, the parameters for the encrypted columns are passed as varbinary. This means the values are encrypted before they even reach SQL Server.



In SSIS, we can see that the Execute SQL Task has executed successfully. However, we can only insert one row at a time. For performance reasons, the data flow will be a better choice to insert multiple rows into a table with encrypted columns.



Writing data to a table with encrypted columns are straight forward: you have to import the certificate and you need to specify a connection property in the ADO.NET connection manager, just like when you want to read data. Insert data through an Execute SQL Task is a bit more complicated, as parameters must be used. This means you can only insert one record at a time.

# SSMS Encryption Wizard – Enabling Always Encrypted in a Few Easy Steps

From Microsoft:

To start with the new wizard (and to follow the examples presented in this article), you will need:

* SQL Server Management Studio [October 2015 preview – version 13.0.700.242](https://msdn.microsoft.com/en-us/library/mt238290.aspx), or later. You can either obtain SSMS as a [standalone download](https://msdn.microsoft.com/en-us/library/mt238290.aspx) as part of [SQL Server 2016](http://www.microsoft.com/en-us/server-cloud/products/sql-server-2016/) Community Technology Preview (CTP) 3.0, or later.
* A database in Azure SQL Database V12 or in a [SQL Server 2016](http://www.microsoft.com/en-us/server-cloud/products/sql-server-2016/) (CTP3 or later) instance.

Note: Until CTP3/October 2015 refresh of SSMS, encrypting existing data either required writing custom code or using the SQL Server Import/Export Wizard, which we described in the [Encrypting Existing Data with Always Encrypted](http://blogs.msdn.com/b/sqlsecurity/archive/2015/07/28/encrypting-existing-data-with-always-encrypted.aspx) article. While the Import/Export Wizard continues to be a useful tool when you need to migrate your data from a plaintext version of a table/database to a different table/table using Always Encrypted, the new Encrypt Columns wizard is now the recommended approach to enable Always Encrypted and encrypt data in an existing database/table.

# Sample Database

In the following examples, we assume our database contains the following table, named Patients, defined below.

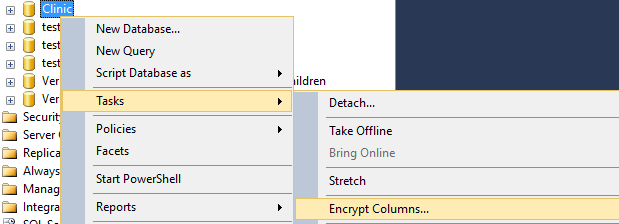
CREATE TABLE [dbo].[Patients](  
 [PatientId] [int] IDENTITY(1,1) NOT NULL,  
 [SSN] [nvarchar](11) NOT NULL,  
 [FirstName] [nvarchar](50) NOT NULL,  
 [LastName] [nvarchar](50) NOT NULL,  
 [MiddleName] [nvarchar](50) NULL,  
 [StreetAddress] [nvarchar](50) NULL,  
 [City] [nvarchar](50) NULL,  
 [ZipCode] [int] NULL,  
 [State] [nvarchar](50) NULL,  
 [BirthDate] [datetime2](7) NOT NULL,  
 PRIMARY KEY CLUSTERED   
 (  
 [PatientId] ASC  
 ) WITH (  
 PAD\_INDEX = OFF, STATISTICS\_NORECOMPUTE = OFF, IGNORE\_DUP\_KEY = OFF,   
 ALLOW\_ROW\_LOCKS = ON, ALLOW\_PAGE\_LOCKS = ON) ON [PRIMARY]  
 ) ON [PRIMARY]

# Encrypt Column Wizard Steps

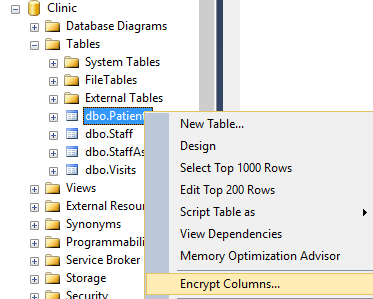
## Starting the Wizard

You can launch the wizard from multiple entry points:

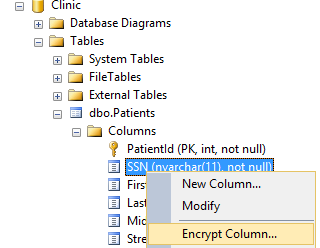
* To encrypt columns located in multiple tables, right-click your database in Object Explorer and select Tasks>Encrypt Columns….

[](https://msdnshared.blob.core.windows.net/media/MSDNBlogsFS/prod.evol.blogs.msdn.com/CommunityServer.Blogs.Components.WeblogFiles/00/00/00/92/93/8551.1.png)

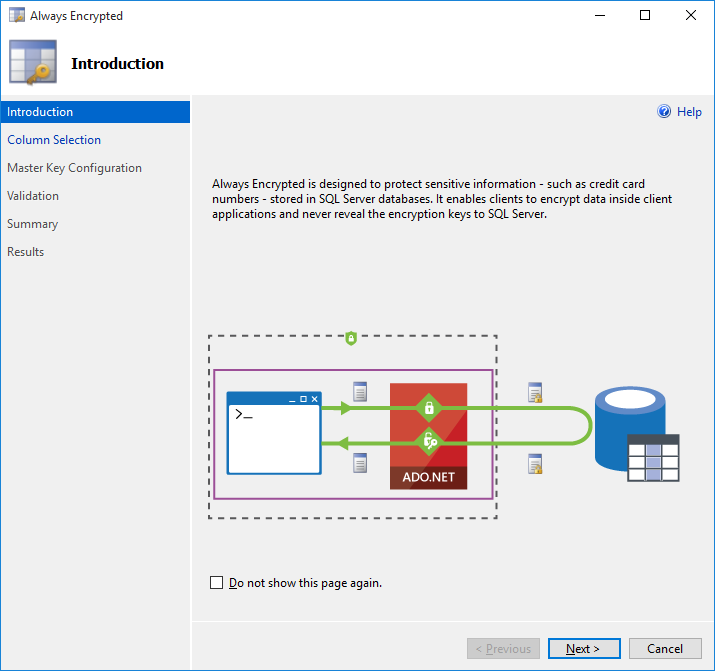
* To encrypt columns located in one particular table, navigate to your table in Object Explorer, right click on it and select Encrypt Columns….

[](https://msdnshared.blob.core.windows.net/media/MSDNBlogsFS/prod.evol.blogs.msdn.com/CommunityServer.Blogs.Components.WeblogFiles/00/00/00/92/93/5707.2.png)

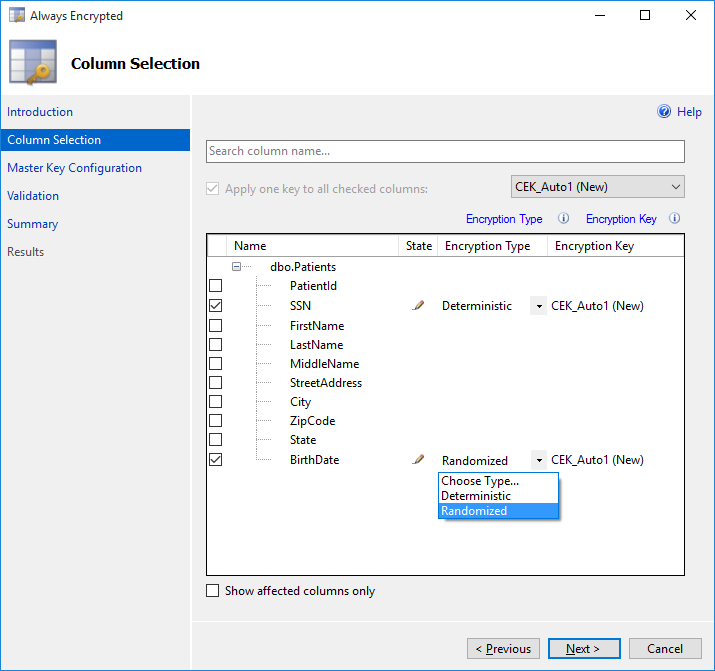
* To encrypt one particular column, navigate to the column in Object Explorer, right click on the column and select Encrypt Column….

[](https://msdnshared.blob.core.windows.net/media/MSDNBlogsFS/prod.evol.blogs.msdn.com/CommunityServer.Blogs.Components.WeblogFiles/00/00/00/92/93/1145.3.png)

Once the wizard starts, the Introduction page should open. Click Next > to go to the next step.

[](https://msdnshared.blob.core.windows.net/media/MSDNBlogsFS/prod.evol.blogs.msdn.com/CommunityServer.Blogs.Components.WeblogFiles/00/00/00/92/93/2043.4.png)

# Column Selection

[](https://msdnshared.blob.core.windows.net/media/MSDNBlogsFS/prod.evol.blogs.msdn.com/CommunityServer.Blogs.Components.WeblogFiles/00/00/00/92/93/4527.5.png)

On the Column Selection page, select columns you want to encrypt using the checkboxes next to column names.

Note: Always Encrypted is not supported for columns using certain data types (e.g. XML, CRL types, aliases, etc.) or certain features (e.g. columns with check constraints, statistics, etc.). For the complete list of limitations, see [Always  
Encrypted (Database Engine)](https://msdn.microsoft.com/en-us/library/mt163865.aspx). In its current version, the wizard checks your schema against a subset of unsupported conditions. For example, the wizard will not let you select a column that uses an unsupported data type. However, the validation against the complete list of limitations is only performed when you complete the wizard steps and trigger the actual encryption. In one of the next releases of SSMS, we will enhance the logic in the Column Selection page to ensure the validation is more complete.

Next, for each selected column, you need to pick an encryption type: Deterministic or Randomized.

* Deterministic encryption supports equality comparison on encrypted columns, i.e. the database system can infer whether two plaintext values are equal by comparing their ciphertext values. Deterministic encryption supports grouping, filtering (exact match) and equality joins, but it can also enable an unauthorized user to guess plain text values by examining patterns in encrypted values.
* Randomized encryption generates a different ciphertext value for the same plaintext, so it is more secure, but it also prevents any T-SQL operations on encrypted columns and indexing.

You will also need to choose a column encryption key for each column to be encrypted. As we discussed in [previous articles](http://blogs.msdn.com/b/sqlsecurity/archive/2015/06/04/getting-started-with-always-encrypted.aspx), Always Encrypted uses two types of keys:

* Column encryption keys – used to encrypt data in your database columns.
* Column master keys – key-protecting keys used to encrypt column encryption keys.

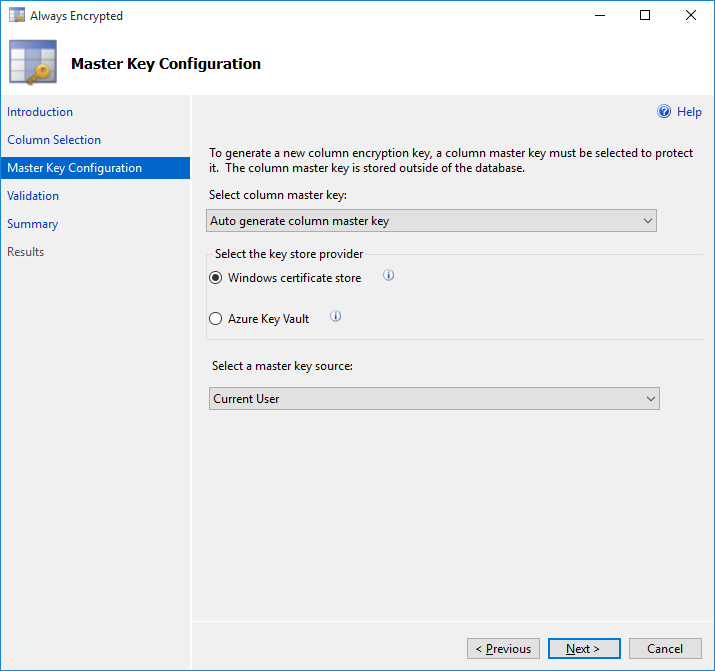
In this step of the wizard, you need to specify a column encryption key for the columns you are encrypting. If no column encryption key is defined in your database yet, the default value in the Encryption Key column in the Column Selection page is CEK\_Auto1(New), which indicates the wizard will generate a new column encryption key named CEK\_Auto1.

Click Next > to go proceed to the next step.

## Best Practice

* Keep in mind that once a column is encrypted, T-SQL functionality for the columns becomes restricted (no operations with randomized encryption, only equality comparison with deterministic encryption). Therefore, before enabling encryption in a production database, you should encrypt columns in a test database first, and carefully test your application workload to identify queries that are impacted by encryption. In general, it is recommended to start by encrypting a small subset of columns that contain the most sensitive information that requires the highest level of protection.
* Use deterministic encryption if your application need to perform exact match searches, equality joins or group on the selected column AND if the set of values, stored in the column, is large and randomly distributed.
* Use randomized encryption if your application does not perform operations on encrypted data on the column, or if the column contains few values (e.g. true/false, gender) or if the distribution of values stored in the column is publically known (e.g. race of medical patients or college students).

# Master Key Configuration

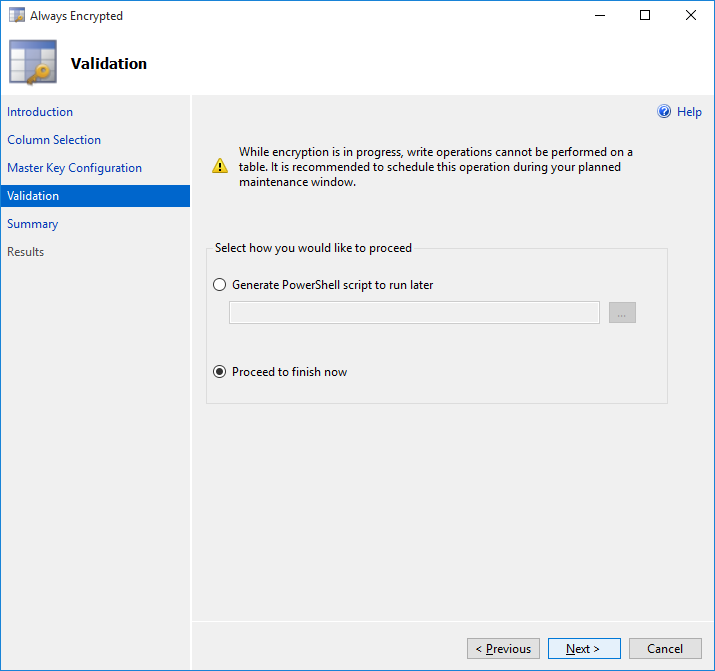
[](https://msdnshared.blob.core.windows.net/media/MSDNBlogsFS/prod.evol.blogs.msdn.com/CommunityServer.Blogs.Components.WeblogFiles/00/00/00/92/93/5758.6.png)

If you chose an auto-generated column encryption key in the previous step, you now need to configure its column master key. If you already have a column master key setup in your database, simply select it. If not, you can use the wizard to also generate the column master key. For that, specify where your column master key is to be stored. You have a couple of options:

* [Windows certificate store](https://msdn.microsoft.com/en-us/library/windows/hardware/ff548653(v=vs.85).aspx%20for%20more%20details.). If you pick this option, the wizard will generate a self-signed certificate and will put it in Windows certificate store on the machine, where SSMS is running. You will need to set the master key source to one of the following:
  + Current User – the generated certificate will be stored in your user-specific certificate store location.
  + Local Machine – the generated certificate will be stored in the global certificate store location that is shared by multiple users on the machine. Note: to create certificates in the Location Machine store location, you need to be running SSMS as an administrator
* [Azure Key Vault](https://azure.microsoft.com/en-us/services/key-vault/) – an Azure cloud service (currently in preview) for storing secrets, including cryptographic keys. If you choose this option, you will be prompted to sign in to Azure, and select an existing vault in Azure Key Vault. See [The Official Azure Key Vault Team Blog](http://blogs.technet.com/b/kv/archive/2015/01/09/azure-key-vault-step-by-step.aspx) as well as our preview article [Creating Custom Key Store Providers for Always Encrypted (Azure Key Vault Example)](http://blogs.msdn.com/b/sqlsecurity/archive/2015/09/25/creating-an-ad-hoc-always-encrypted-provider-using-azure-key-vault.aspx) for how to create and set up a vault in Azure Key Vault.

Click Next > to proceed to the next step.

# Generate PowerShell or Proceed with Encryption

[](https://msdnshared.blob.core.windows.net/media/MSDNBlogsFS/prod.evol.blogs.msdn.com/CommunityServer.Blogs.Components.WeblogFiles/00/00/00/92/93/4571.8.png)

In this step, you can either choose to save the encryption workflow as a PowerShell script to be executed later, or to proceed with encryption now.

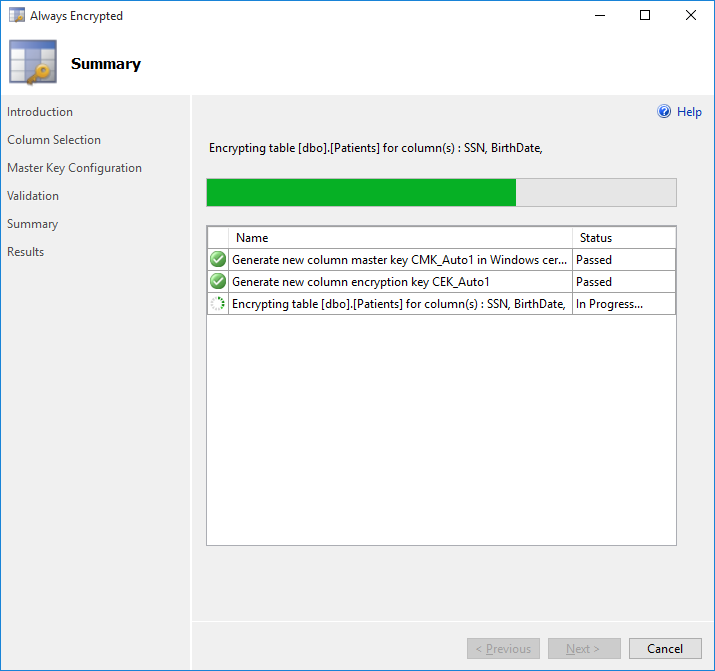
Note: When using the current version of the wizard, you need to make sure no other application inserts or updates rows in the tables, containing encrypted columns, while the encryption workflow is running. During the encryption workflow, the wizard creates a temporary table, downloads the data from your original table, encrypts the data and uploads it to the temporary table. Finally, the wizard deletes the original table and renames the temporary table to the original table. If another app is inserting or modifying data in the original table, the new or updated data may be lost. Make sure, you only run the encryption workflow in a planned maintenance window. This issue will be addressed in a later version of SSMS.

## Best Practice

In general, it is recommended you create a database backup before running the wizard.

# Reviewing Workflow Steps

 Examine the encryption steps and click Finish to trigger the workflow. Monitor progress of encryption and then close the wizard.

[](https://msdnshared.blob.core.windows.net/media/MSDNBlogsFS/prod.evol.blogs.msdn.com/CommunityServer.Blogs.Components.WeblogFiles/00/00/00/92/93/4861.10.png)

# Explore the Results

Let us examine the results of running the wizard:

* Table schema changes,
* New key metadata,
* Generated key in a column master key store,
* Encrypted data.

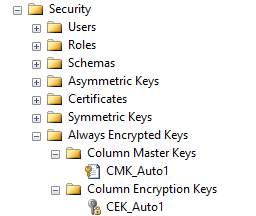
# Table Schema Changes

If you script the schema of the table, containing columns you have encrypted, you will notice the ENRYPTED WITH clause has been added to the encrypted columns.

CREATE TABLE [dbo].[Patients](  
 [PatientId] [int] IDENTITY(1,1) NOT NULL,  
 [SSN] [nvarchar](11) ENCRYPTED WITH (COLUMN\_ENCRYPTION\_KEY = [CEK\_Auto1], ENCRYPTION\_TYPE = Deterministic, ALGORITHM = 'AEAD\_AES\_256\_CBC\_HMAC\_SHA\_256') NOT NULL,  
 [FirstName] [nvarchar](50) NOT NULL,  
 [LastName] [nvarchar](50) NOT NULL,  
 [MiddleName] [nvarchar](50) NULL,  
 [StreetAddress] [nvarchar](50) NULL,  
 [City] [nvarchar](50) NULL,  
 [ZipCode] [int] NULL,  
 [State] [nvarchar](50) NULL,  
 [BirthDate] [datetime2](7) ENCRYPTED WITH (COLUMN\_ENCRYPTION\_KEY = [CEK\_Auto1], ENCRYPTION\_TYPE = Randomized, ALGORITHM = 'AEAD\_AES\_256\_CBC\_HMAC\_SHA\_256') NOT NULL,  
 PRIMARY KEY CLUSTERED   
 (  
 [PatientId] ASC  
 )WITH (PAD\_INDEX = OFF, STATISTICS\_NORECOMPUTE = OFF, IGNORE\_DUP\_KEY = OFF, ALLOW\_ROW\_LOCKS = ON, ALLOW\_PAGE\_LOCKS = ON) ON [PRIMARY]  
 ) ON [PRIMARY]

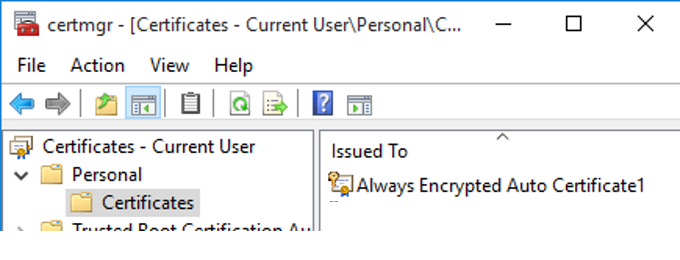
# New Key Metadata

In Object Explorer, navigate to the Security/AlwaysEncrypted Keys folder under your database. There, you will find entries for both the column master key and the column encryption key, the wizard generated.

[](https://msdnshared.blob.core.windows.net/media/MSDNBlogsFS/prod.evol.blogs.msdn.com/CommunityServer.Blogs.Components.WeblogFiles/00/00/00/92/93/8203.11.png)

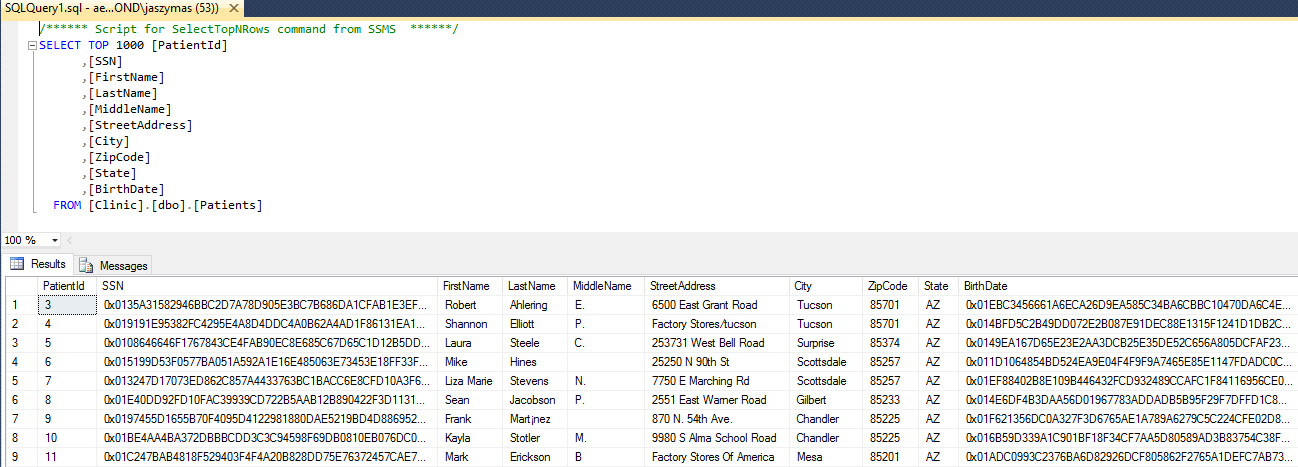
# Generated Key

If you chose to generate a self-signed certificate to be used a column master key, you can find and explore the properties of the certificate. Run certmgr.msc if your certificate is stored in the Current User location, or run certlm.msc if it is stored in the Local Machine location. You will find your certificate under the Personal store.

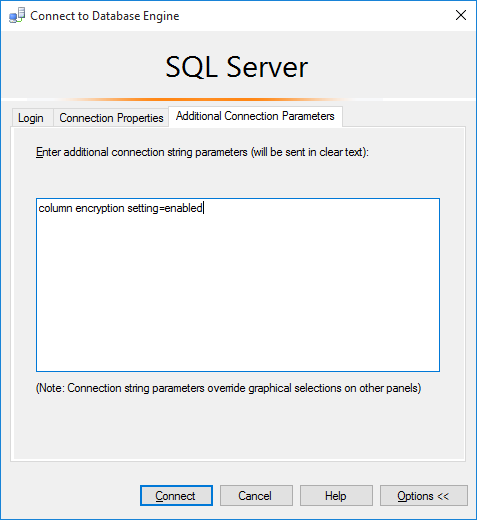
[](https://msdnshared.blob.core.windows.net/media/MSDNBlogsFS/prod.evol.blogs.msdn.com/CommunityServer.Blogs.Components.WeblogFiles/00/00/00/92/93/0523.22.png)

# Encrypted Data

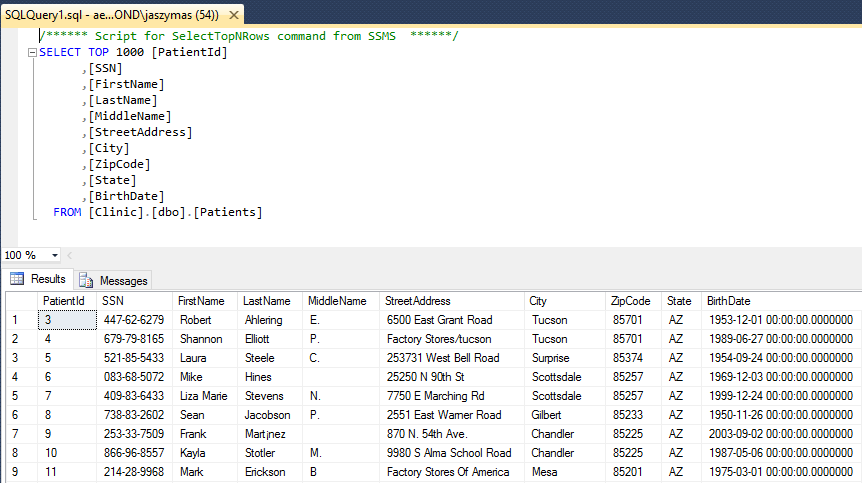
When you retrieve data from your table, you will notice binary encrypted values in the columns you have encrypted.

[](https://msdnshared.blob.core.windows.net/media/MSDNBlogsFS/prod.evol.blogs.msdn.com/CommunityServer.Blogs.Components.WeblogFiles/00/00/00/92/93/3858.13.png)

Now, reconnect to the database by adding the following connection string keyword/value: column encryption setting=enabled:

[](https://msdnshared.blob.core.windows.net/media/MSDNBlogsFS/prod.evol.blogs.msdn.com/CommunityServer.Blogs.Components.WeblogFiles/00/00/00/92/93/8015.15.png)

Re-run the same query using the new connection. This time, the query should return plaintext values from encrypted columns. Why? Because the above connection string keyword/value enables Always Encrypted in the client driver and causes the driver to transparently decrypt the data retrieved from encrypted columns. Since you have an access to the column master key protecting your columns (the key is either in Windows certificate store on your machine or in a vault you can access), decryption succeeds and you can view plaintext data.

[](https://msdnshared.blob.core.windows.net/media/MSDNBlogsFS/prod.evol.blogs.msdn.com/CommunityServer.Blogs.Components.WeblogFiles/00/00/00/92/93/2860.16.png)

To see what happens if you instruct the driver to decrypt query results, but you do not have access to your column master key, perform the following test:

1. If your column master key is:
   * A certificate stored in the Current User location: log on to the same machine using a different account, or log on to another machine.
   * A certificate stored in the Local Machine location: log on to another machine.
   * A key in Azure Key Vault: log on to the same or a different machine as another user, who does not have a permission in Azure Key Vault to access the key.
2. Start SSMS and connect to your database with column encryption setting=enabled.
3. Run a query to retrieve data from encrypted columns.

Even if you connected to the database as a DBA, you will get an error, indicating you have no access to the column master key and, therefore you will not be able to see the data in plaintext (you can still retrieve ciphertext, if you connect without column encryption setting=enabled). See the previous article, [Always Encrypted Key Metadata](http://blogs.msdn.com/b/sqlsecurity/archive/2015/07/06/always-encrypted-key-metadata.aspx), for details on how the keys and key metadata are used for decryption.

# Next Steps

You now have encrypted sensitive data in your database. Next:

* You can re-run the wizard to encrypt more columns, change any encryption settings, or to decrypt a column you encrypted previously.
* Test your stored procedures, views, and queries inside your applications. You might need to refactor your stored procedures/views/queries if they use operations that are unsupported for the type of encryption you configured for your columns.
* Make sure your client applications or middle-tier services that need access to sensitive data in plaintext, are able to decrypt the data. This will involve:
  + Configuring the applications to use a SQL client driver that supports Always Encrypted.
  + Modifying database connection strings to include column encryption setting=enabled.
  + Making the column master key available to the application. This steps will depend on where you column master key is stored.

If your column master key is a certificate in Windows Certificate Store, you will need to deploy the certificate to each machine hosting your application/service to the right store location (Current User or Local Machine).

# Managing Always Encrypted Certificates

From <https://www.mssqltips.com/sqlservertip/4814/exporting-and-importing-sql-server-always-encrypted-certificates-for-client-access/>

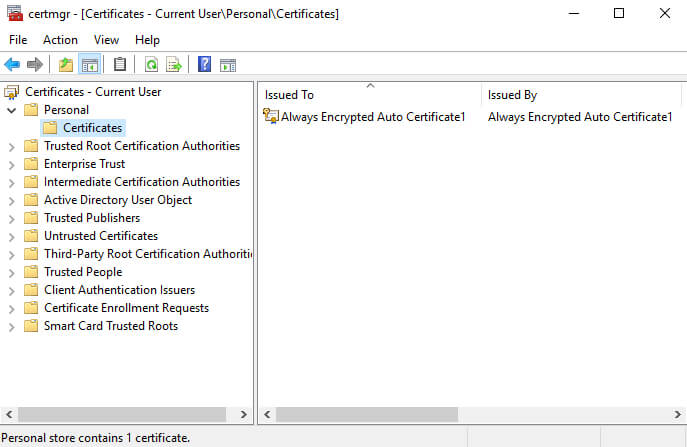
## Exporting the Certificate

After data has been [encrypted using Always Encrypted](https://www.mssqltips.com/sqlservertip/4011/sql-server-2016-always-encrypted/), a certificate is created on the database server. You will need to export this certificate, then import the certificate to the client machine(s) that require access to work with the encrypted data.

To view the list of certificates that are created on the database server you can execute certmgr from a command line:

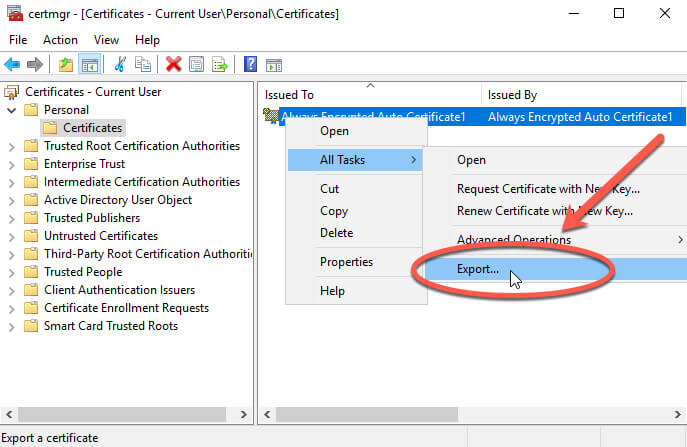


This will launch the certmgr MMC, here's what it looks like on my VM I am using for this post:

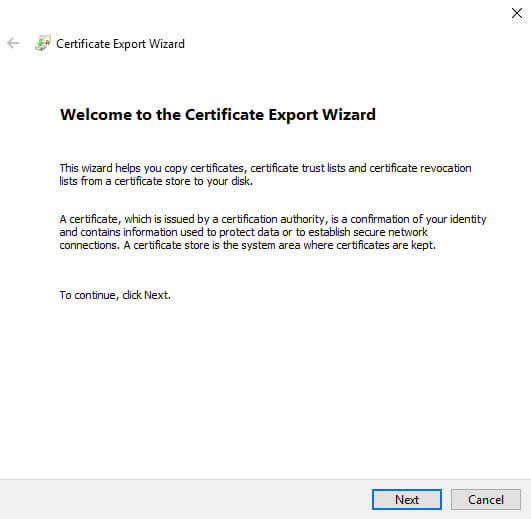


As with the example linked above, the Always Encrypted certificate was created as the current user, and it can be found in the Personal folder. You can also do a search for "Always Encrypted" to locate the certificate(s) created on the database server.

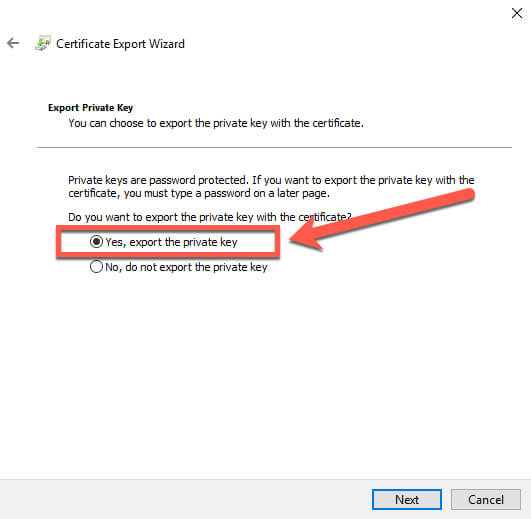
To export the certificate we will do a right-click, select "All Tasks", then export:



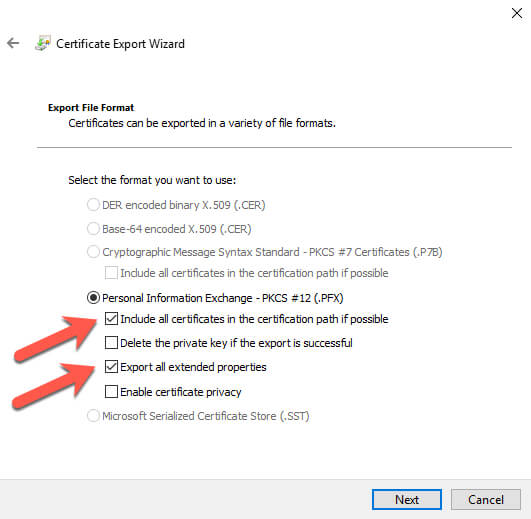
This launches the Certificate Export Wizard:



We will click next, and you arrive at the most important screen in the export process:

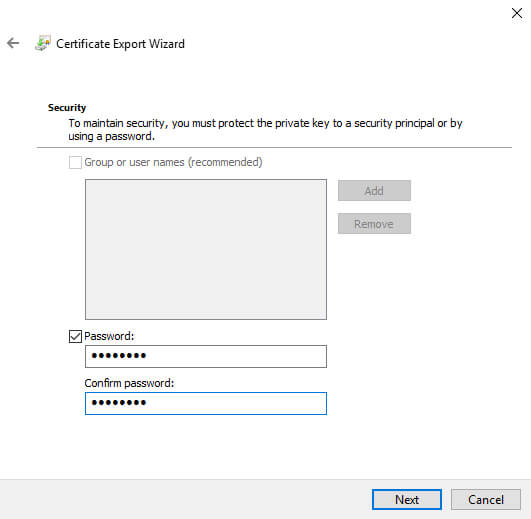


You must select the "Yes, export the private key" option. If you do not select this option, then the certificate will be worthless upon import. We select the option, then click Next, and we have more options:

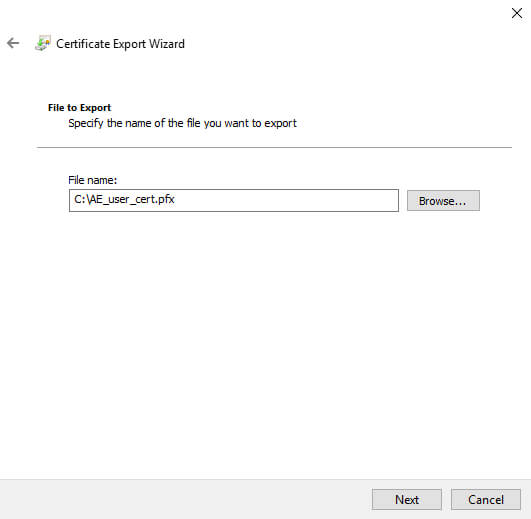


I choose the "Include all certificates in the certification path if possible" and "Export all extended properties" options. I will not delete the private key at this time. Depending on your requirements, you may want to remove the key later, but I would advocate that you verify the import works correctly before removing the private key. Otherwise you are going to be stuck with encrypted data and you will need to recover from backups and start the encryption process again.

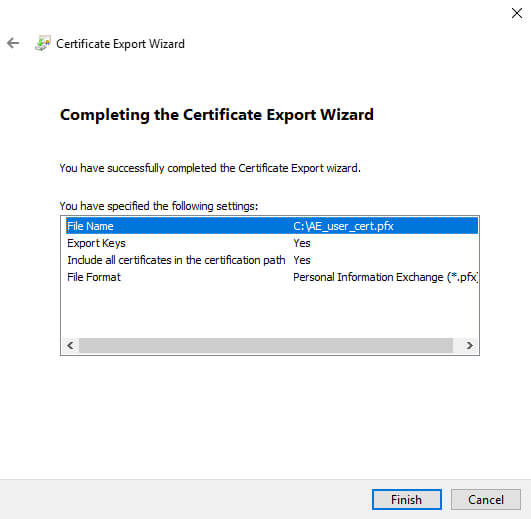
We click Next and will be prompted to set a password:



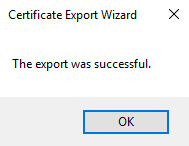
We choose a strong password then click Next:



We choose a path and filename, then click Next:



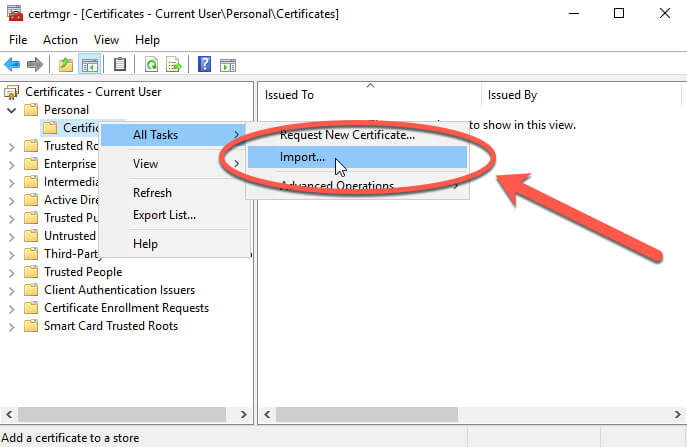
Review the details and click Finish. You should then see this:



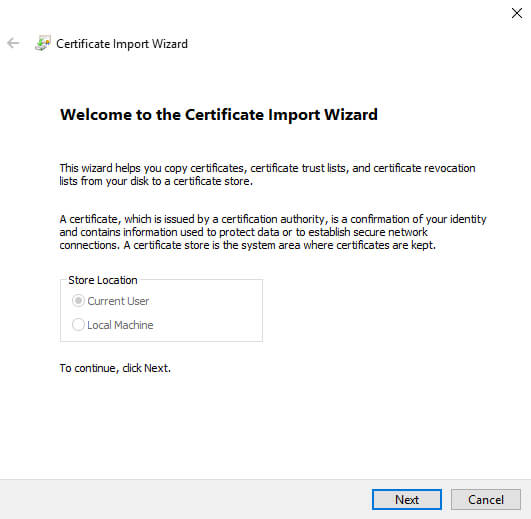
Now that we have exported the certificate, let's try to do the import to a client machine and verify that we can view the decrypted data. We start the import process by opening certmgr again.

## Importing the Certificate

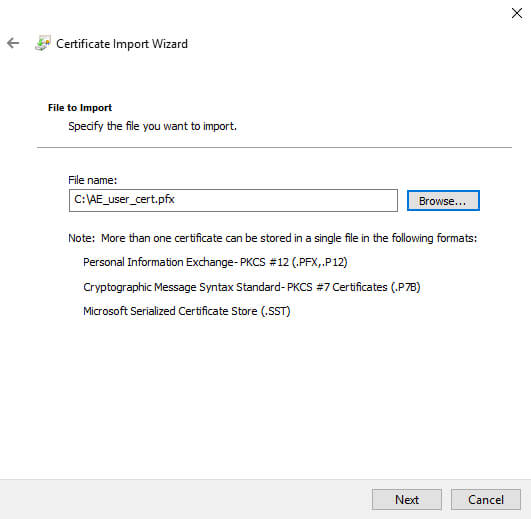
Navigate to the certificates folder in the personal store, and right click to start the import process:



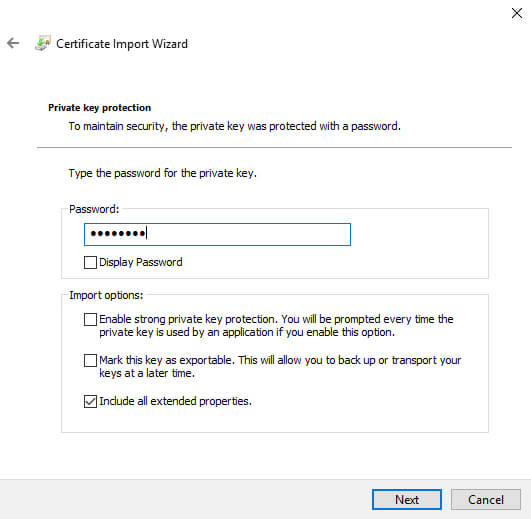
This will launch the Certificate Import Wizard, and you can see that the current user is already chosen:



We will click next, and navigate to the file we exported:

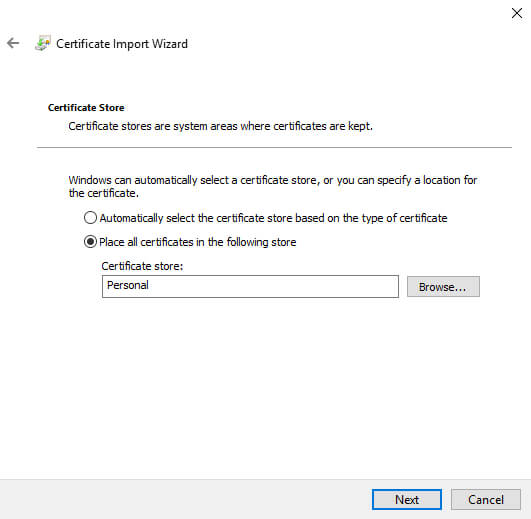


We click next, and enter the strong password used in the export process:

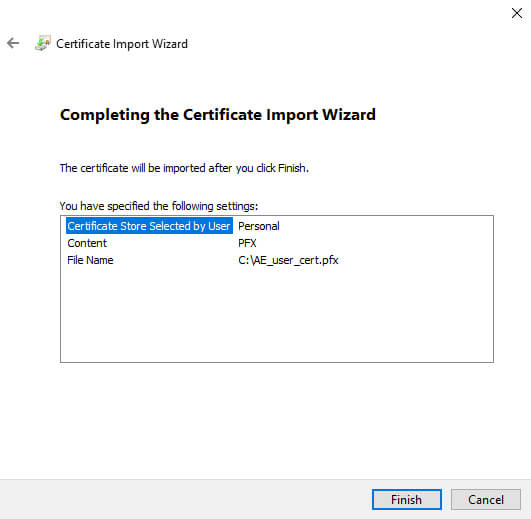


I chose to include the extended properties, but nothing else. Note that I am not allowing this key to be exportable. More on that later.

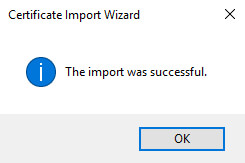
We now click next:



The Personal store is already chosen, that's the option we are using in this example, so we click Next:



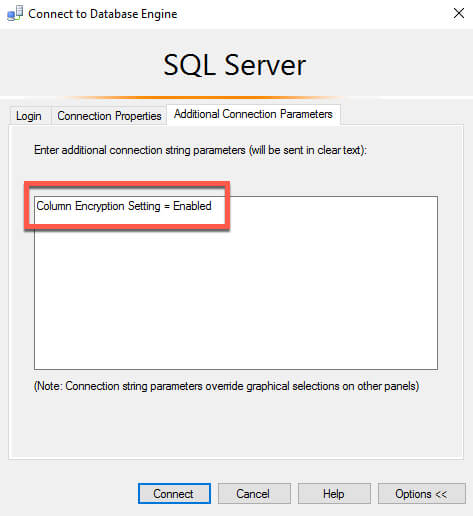
Review the details, then click Finish, and we should see:



Next, we want to verify the import is working as expected.

## Verify import was successful

The permissions are already in place, so we will next update our connection string. We can do that inside of SSMS, in the advanced properties for the connection. We want to add in the string "Column Encryption Setting = Enabled", like this:



Once we update our connection to be enabled for encryption, we can run our query again and view the unencrypted data.

# Example Demonstration – Modifying an Existing Application

# From Codeproject.com is a complete end to end demonstration including how to reconfigure an existing application. The article is by [Manjuke Fernando](https://www.codeproject.com/script/Membership/View.aspx?mid=380840) from https://www.codeproject.com/Articles/1142885/Always-Encrypted-feature-in-SQL-Server

## Always Encrypted Demonstration

We will see how Always Encrypted can be implemented and used. In order to illustrate, we will use a table which contains employee information.

Hide   Shrink https://www.codeproject.com/images/arrow-up-16.png   Copy Code

CREATE TABLE Employee(

Id INT

,FirstName VARCHAR(100)

,LastName VARCHAR(100)

,DOB DATE

,SSN INT

,[Address] VARCHAR(255)

,PostalCode INT

)

INSERT INTO Employee (

[Id],[FirstName],[LastName],[DOB],[SSN],[Address],[PostalCode])

VALUES

(1,'James','Rubin','20-Jul-1986',173456858,'10585 N 600 E',46310)

,(2,'Austin','Pyatt','24-Dec-1985',138868248,'100 BENTBROOK CT',27519)

,(3,'Stacey','Munoz','23-Dec-1988',185682639,'1 WOODSIDE DR',4976)

,(4,'James','Tweed','03-Jan-1987',133890886,'1 AUNNEK CT',95023)

,(5,'James','Robles','11-Sep-1989',154135505,'101 FISHTRAP RD',35504)

,(6,'Ebony','Lewis','17-Jul-1988',120488337,'101 N OAKS DR',35180)

,(7,'Marian','Caro','20-Nov-1985',115281829,'1017 FISK ST SE',49507)

,(8,'Lynne','Martinez','22-Apr-1985',157900240,'103 UNITY CT',78214)

,(9,'Elsa','Cole','25-Apr-1990',150631885,'1001 E FERN AVE APT 201',78501)

,(10,'Kiley','Caldwell','03-Jan-1988',131368172,'103 NOB HILL LN APT 5',40206)

,(11,'Michael','Soluri','17-Jun-1985',173245124,'10770 S KILBOURN AVE',60453)

,(12,'Gregory','Emmons','06-Sep-1988',137693229,'10 LOUISA PL APT 2F',7086)

,(13,'Jessica','Barr','04-Feb-1989',155895863,'1 FAWNRIDGE DR',94945)

,(14,'Daniel','Mccabe','06-Sep-1985',148236776,'1 CALLE MARGINAL GARCIA',674)

,(15,'Sharon','Schwartz','06-Sep-1987',117569460,'1 KRITTER CT',8050)

,(16,'Dorthy','Wear','13-Dec-1988',170517705,'1 CLARK RD',35747)

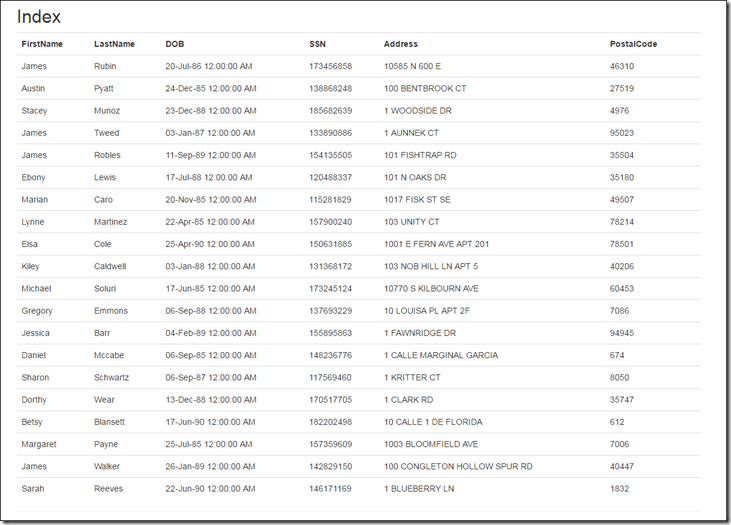
,(17,'Betsy','Blansett','17-Jun-1990',182202498,'10 CALLE 1 DE FLORIDA',612)

,(18,'Margaret','Payne','25-Jul-1985',157359609,'1003 BLOOMFIELD AVE',7006)

,(19,'James','Walker','26-Jan-1989',142829150,'100 CONGLETON HOLLOW SPUR RD',40447)

,(20,'Sarah','Reeves','22-Jun-1990',146171169,'1 BLUEBERRY LN',1832)

I have a small MVC Web Application which has a page to list out the aforementioned details from the SQL Server. The MVC Controller will load the details to a list of Employee records and pass it to the Html view which will be displayed as follows.

[](https://lh3.googleusercontent.com/-IpPhR7rWJJw/WAav73N-IVI/AAAAAAAAEC8/r4Dj4ryaFgs/s1600-h/image4.png)

In the MVC application I have the following data model to load details from the SQL Database Table.

Hide   Copy Code

public class Employee {

public int Id { get; set; }

public string FirstName { get; set; }

public string LastName { get; set; }

public DateTime DOB { get; set; }

public int SSN { get; set; }

public string Address { get; set; }

public int PostalCode { get; set; }

public Employee() {

}

}

And I am using the following connection string in order to connect to the SQL Server Database.

Hide   Copy Code

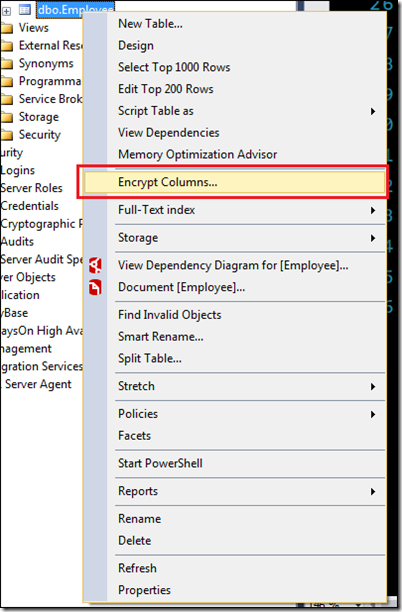
const string zConnectionString =

@"Server=.\SQL2K16; Network Library=DBMSSOCN;Database=SQLTraining;Trusted\_Connection=True;";

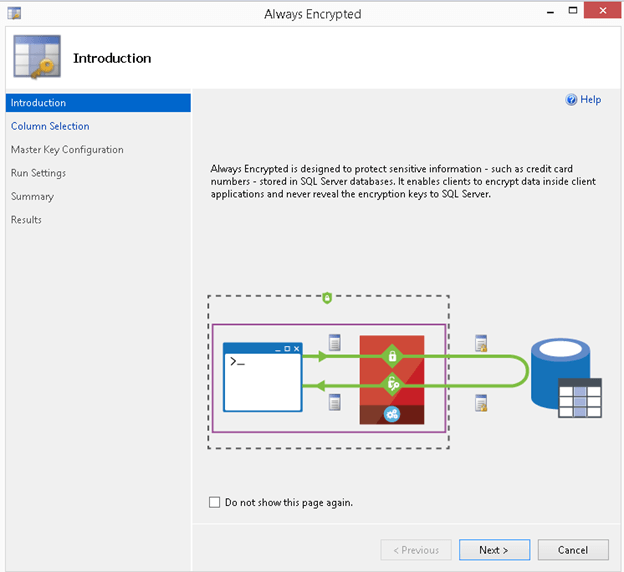
There are few steps to be followed on both SQL Server and application side (Client Applications) in order to implement and use this feature.

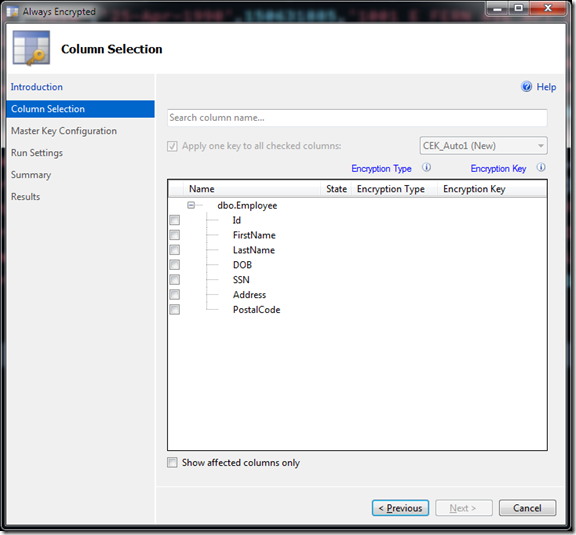
From the SQL Server side, there are few ways to enable the Always Encrypted feature. We will look more details how to use these feature using the wizard.

1. Right click the table which you want to encrypt details and select ‘Encrypt Columns’. This will take you to the wizard.

[](https://lh3.googleusercontent.com/-beZ1L5sfE8U/WAX7qYorf8I/AAAAAAAAECA/gnjVCqvUuYw/s1600-h/image%5b8%5d.png)

2. You will get the introduction screen which contains few details about what ‘Always Encrypted’ is all about. Click next and proceed to the next screen.



[](https://lh3.googleusercontent.com/-EJLn1-8L_lM/WAX7rQsG9AI/AAAAAAAAECI/6PecJwTmtic/s1600-h/image%5b12%5d.png)

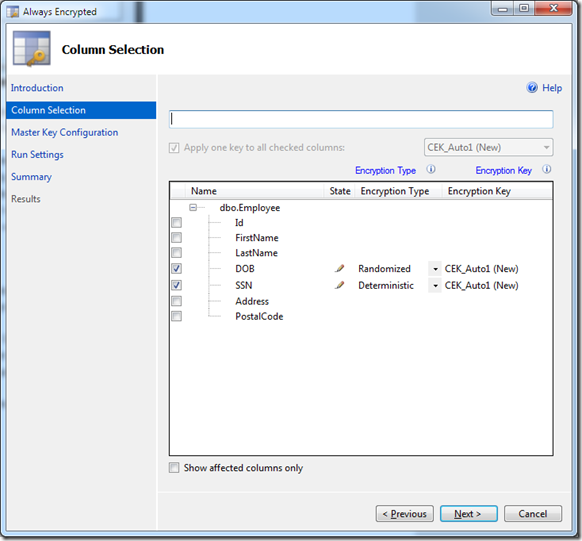
This is the column selection screen, which allows you to select which columns you want to encrypt and using which Encryption Type. There are two Encryption Types available in SQL Server 2016.

* **Deterministic** –> 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** –> 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.

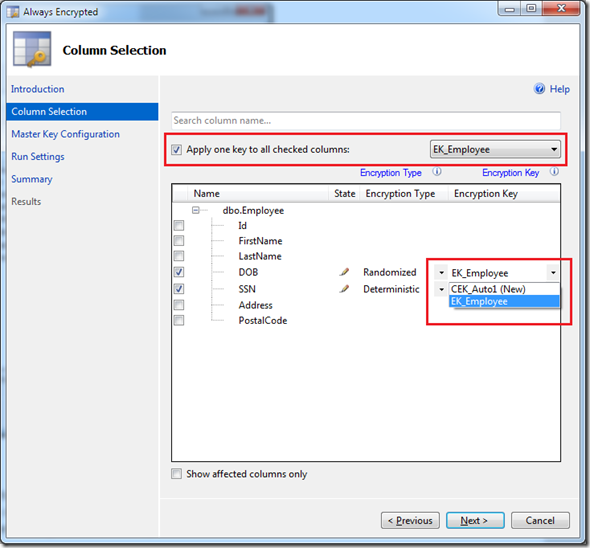
This advice has been included in Microsoft Documentation: Use deterministic encryption for columns that will be used as search or grouping parameters, for example a government ID number. Use randomized encryption, for data such as confidential investigation comments, which are not grouped with other records and are not used to join tables.

So in our example we will choose DOB & SSN columns for encryption. For DOB we will choose Randomized and for SSN we will choose Deterministic.

Once the encryption type is chosen the wizard should be similar to the screen shown below.

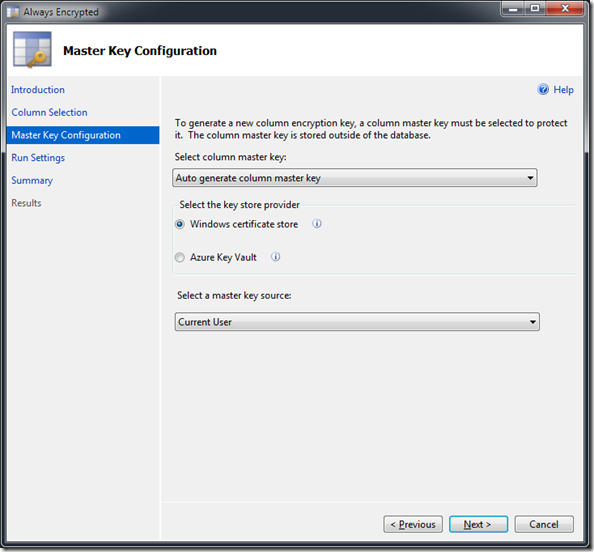
[](https://lh3.googleusercontent.com/-VdbFUZtx2no/WAX7sb5Lv1I/AAAAAAAAECQ/wWR1QfULFmI/s1600-h/image%5b18%5d.png)

If you look closely, you will be able to see that the Encryption Key combo is disabled. The reason for this is the fact that we haven’t created any Column encryption keys so far. If the keys are created prior to the column selection then you will have the option to choose whether to use an existing key or to generate a new key.

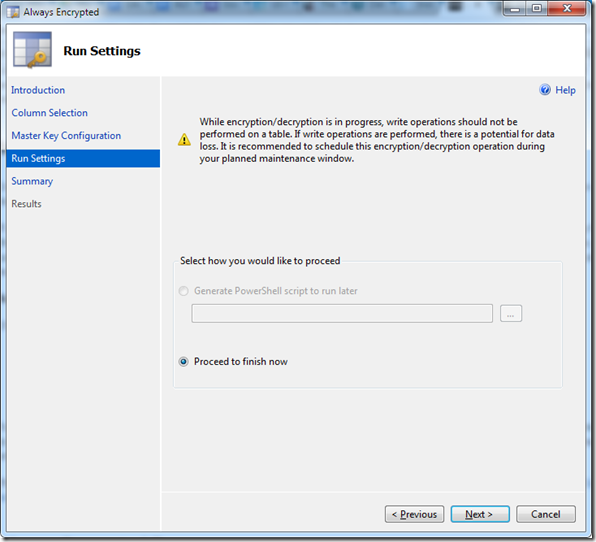
[](https://lh3.googleusercontent.com/-g-r14QHhmOQ/WAX7tfsAaKI/AAAAAAAAECY/gRDW2B2yw3s/s1600-h/image%5b22%5d.png)

In this illustration, we will use the option which will create a new column encryption key. Click next to proceed to the next step.

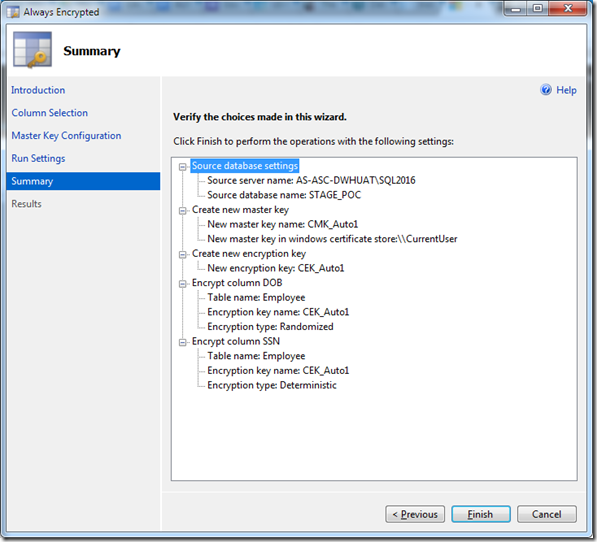
3. The next step is the Column Master Key Configuration. A Column Master Key will be used to encrypt and protect the Column Encryption Key, which is used to encrypt the data. We will use the option ‘Auto generated column master key’, which the wizard will generate the key for us. When we are creating a new Master Key, there are two options available, where to store the newly generated key. Clicking on the small info button beside each option will give further details about each option

[](https://lh3.googleusercontent.com/-FbU87zuaakY/WAav9KNiVQI/AAAAAAAAEDE/guHn0PEmGyA/s1600-h/image8.png)

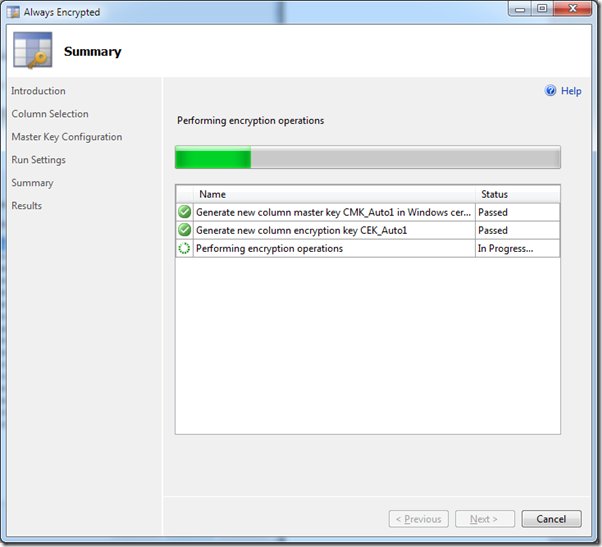
4. Click next to move to the next step. In this step you can decide whether you require a PowerShellscript to be generated for the encryption process or to proceed with the encryption immediately. In this example we will select the second option and click on the next button.

[](https://lh3.googleusercontent.com/-chckVrcicQk/WAav-BwreOI/AAAAAAAAEDM/nHgBezZWA5M/s1600-h/image%5b5%5d.png)

In this step you will be presented with the steps which will be followed during the data encryption

[](https://lh3.googleusercontent.com/-UWmC3i9Q15Q/WAav_A4F5SI/AAAAAAAAEDU/3Q-_LcUwQt0/s1600-h/image%5b10%5d.png)

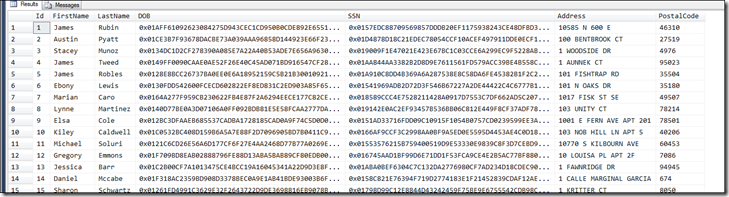
Click finish to complete the encryption process. Once process is completed click close button.

[](https://lh3.googleusercontent.com/-S0GI39Aciwg/WAc5ERc0MuI/AAAAAAAAEDs/Kag52dmWMkg/s1600-h/image%5b4%5d.png)

Now if you check the details on SQL Table you can see that, data in SSN and DOB columns are encrypted.

Hide   Copy Code

SELECT \* FROM dbo.Employee

[](https://lh3.googleusercontent.com/-kN071EP-Zec/WAc5FVKMfII/AAAAAAAAED0/e3Kcz9KDbNk/s1600-h/image%5b8%5d.png)

If you see the Table creation script for the Employee table now, you could see few changes which has been done by the SQL Server after we enabled the encryption for those two columns.

Hide   Copy Code

CREATE TABLE [dbo].[Employee](

[Id] [INT] NULL,

[FirstName] [VARCHAR](100) NULL,

[LastName] [VARCHAR](100) NULL,

[DOB] [DATE] ENCRYPTED WITH (COLUMN\_ENCRYPTION\_KEY = [CEK\_Auto1],

ENCRYPTION\_TYPE = RANDOMIZED,

ALGORITHM = 'AEAD\_AES\_256\_CBC\_HMAC\_SHA\_256') NULL,

[SSN] [INT] ENCRYPTED WITH (COLUMN\_ENCRYPTION\_KEY = [CEK\_Auto1],

ENCRYPTION\_TYPE = DETERMINISTIC,

ALGORITHM = 'AEAD\_AES\_256\_CBC\_HMAC\_SHA\_256') NULL,

[Address] [VARCHAR](255) NULL,

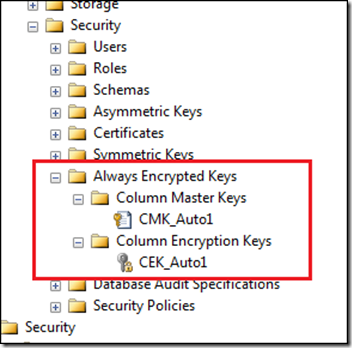
[PostalCode] [INT] NULL

) ON [PRIMARY]

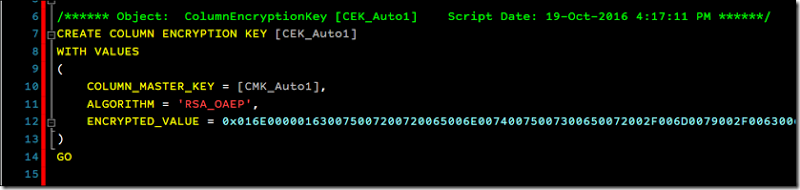
You can see that it had added the ENCRYPTED WITH clause for those two columns. ENCRYPTED WITH clause consist 3 attributes which are:

* COLUMN\_ENCRYPTION\_KEY –> CEK\_Auto1 since we have chosen the option for SQL to generate a new key.
* ENCRYPTION\_TYPE –> Can be either RANDOMIZED or DETERMINISTIC
* ALGORITHM –> This is always AES\_256

If you inspect the Always Encrypted keys in the object explorer in SSMS you could see the following meta data for the Master and the Column Encrypted Keys.

[](https://lh3.googleusercontent.com/-d9MSU8VAh-I/WAc5GfBbS4I/AAAAAAAAED8/P3xwnel5L8I/s1600-h/image%5b12%5d.png)

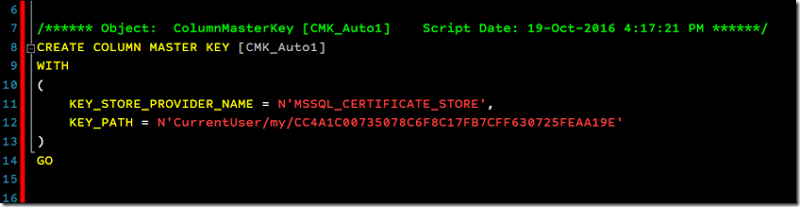
**Column Encrypted Key – CEK\_Auto1**

[](https://lh3.googleusercontent.com/-B05Nd-0hT60/WAc5HVeW57I/AAAAAAAAEEE/3Kv7i29GC5Q/s1600-h/image%5b16%5d.png)

* **COLUMN\_MASTER\_KEY** –> Name of the column master key protecting the value of the column encryption key.
* **ALGORITHM** –> Algorithm used to generate the encrypted value of the column encryption key (RSA\_OAEP).
* **ENCRYPTED\_VALUE** –> Encrypted value of the column encryption key. The encrypted value is assumed to be produced by encrypting the plaintext of the column encryption key using the specified column master key and the specified algorithm.

For further information please refer to the following url: <https://blogs.msdn.microsoft.com/sqlsecurity/2015/07/06/always-encrypted-key-metadata/>

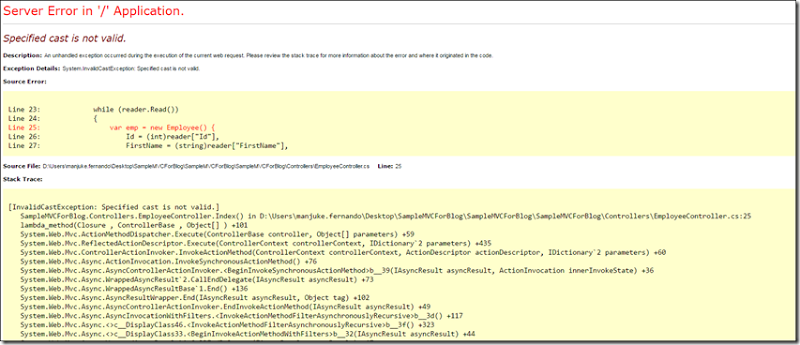
**Column Master Key - CMK\_Auto1**

[](https://lh3.googleusercontent.com/-nPm0rq9cnAc/WAc5IM26YaI/AAAAAAAAEEM/kydPGX5ws68/s1600-h/image%5b20%5d.png)

* **KEY\_STORE\_PROVIDER\_NAME** –> Name of a provider for the key store that holds the column master key.
* **KEY\_PATH** –> Key path specifying the location of the column master key in the key store.

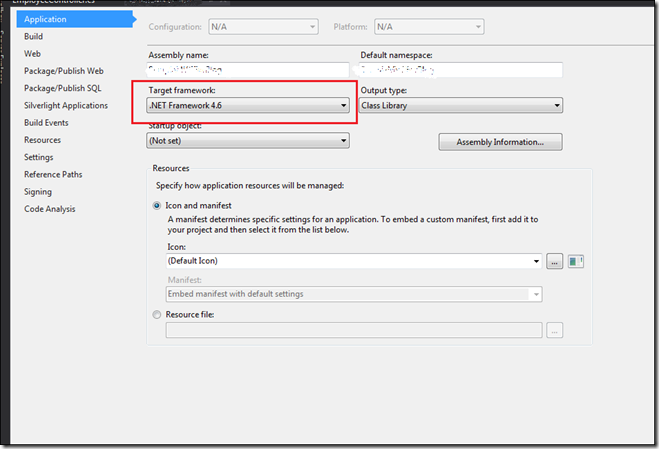
For further information please refer to the following url: <https://blogs.msdn.microsoft.com/sqlsecurity/2015/07/06/always-encrypted-key-metadata/>

Now if we try to fetch details without doing anything on the sample .Net Application you will get a similar error like shown below.

[](https://lh3.googleusercontent.com/-8lha690qQg4/WAc5JFAyQsI/AAAAAAAAEEU/69XLXrgxLwk/s1600-h/image%5b24%5d.png)

Now we will look into the things that we required to change on our application side (Business) in order to retrieve the required information.

**1. Make sure that the target framework is version 4.6 or higher.**

[](https://lh3.googleusercontent.com/-taHyuqjlh0g/WAc5KBiqlrI/AAAAAAAAEEc/DW8pKsTLa4k/s1600-h/image%5b28%5d.png)

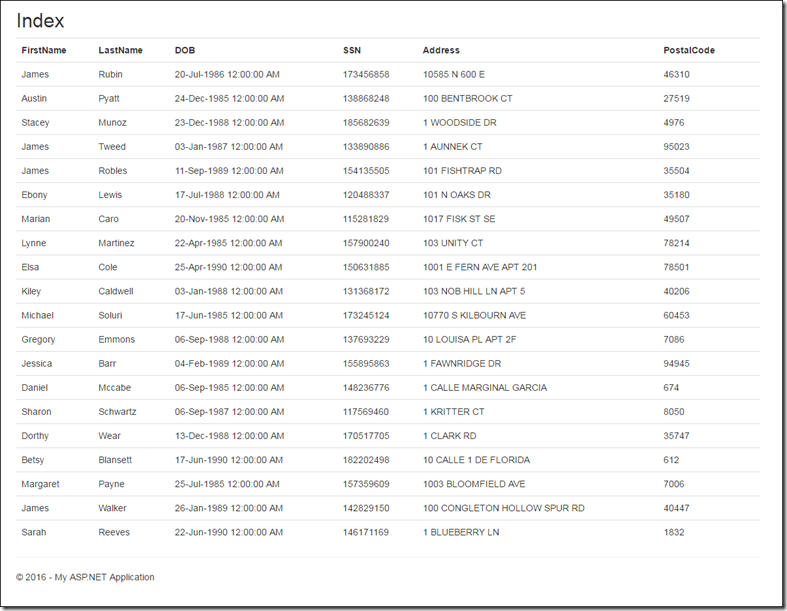
**2. In the Connection String include ‘Column Encryption Setting=enabled’**

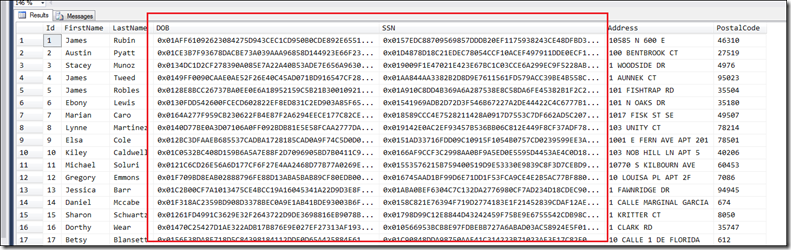
And I am using the following connection string in order to connect to the SQL Server Database.

Hide   Copy Code

const string zConnectionString = @"Server=.\SQL2K16; Network Library=DBMSSOCN;Database=SQLTraining;Trusted\_Connection=True;Column Encryption Setting=enabled;";

Now if we check the details from our application we can see that DOB and SSN values are fetched as plain text, even though the values are encrypted in the SQL Server.

[](https://lh3.googleusercontent.com/-lCWM6JjNEmM/WAc5LYa777I/AAAAAAAAEEk/HZtVuj-584k/s1600-h/image%5b34%5d.png)

[](https://lh3.googleusercontent.com/-sjEqff4Iwas/WAc5MprFOkI/AAAAAAAAEEs/Rfjdiv2mr9E/s1600-h/image%5b39%5d.png)