How to configure Always Encrypted in SQL Server 2016 using SSMS, PowerShell and T-SQL

October 2, 2017 by Prashanth Jayaram

In an era of remote storage and retrieval of data, including the cloud, data security plays a vital role, especially since it's vulnerable during the transit. Situations like database backup or copy from or to the cloud, there is always a risk of data exposure to outside world lurking around one corner or the other. We have seen a noticeable surge in the technologies around protection and security of data from the world full of unsafe hands. Efforts are being made to protect data at a very granular level of the encryption hierarchy. Protection of business data cannot be stressed upon more.

One way of inching towards the more secure transmission of data is to enable Always Encrypted on the database. We'll look into the various options we have, including enabling this at granular levels; we'll look at enabling this at the column level.

The Always Encrypted feature was available only on the Enterprise and Developer editions of SQL Server 2016. Later, this feature was made available on all editions, with SQL Server 2016 SP1. Always Encrypted has the ability to encrypt data even at the column level.

There are several ways to configure the Always Encrypted feature:

- Using the Always Encrypted wizard
- Configuring AE using SSMS
- Create Master Key and Encryption Key using T-SQL and enabling encryption
- Configuring Always Encrypted using PowerShell

Overview of the Always Encrypted Feature

Always Encrypted feature is a handshake mechanism used to encrypt and decrypt data. Encryption

here is achieved using certificates, and can be done only by users with access to the relevant certificates. To make a database column Always Encrypted, you must specify the encryption algorithm and the cryptographic keys that are used to protect the data. Always Encrypted needs two keys:

1. Column Encryption Key (CEK)
2. Column Master Key (CMK)

A Column Encryption Key is used to protect and encrypt data in a column. A Column Master Key is used to protect the (one or more) column encryption keys. The information about the Column Master Key is stored in external key stores like:

- Azure Key Vault: A key vault used to safeguard and manage cryptographic keys and secrets used for encryption and decryption of sensitive data within Microsoft Azure.
- Windows Certificate Store: A certificate container built into Windows that stores and manages the certificates.
- Hardware Security Module (HSM): A hardware device specially designed to securely store sensitive data

Selecting Deterministic or Randomized Encryption

Always Encrypted supports two types of encryption: randomized and deterministic

- Deterministic encryption
  - The same encrypted Key for a given value is generated, every time.
  - Binary2 sort order collation must be used to setup deterministic encryption on a column.
  - Heuristically studying the patterns of the contents of the column could reveal the contents, thereby making it more susceptible to hacking
- Randomized encryption
  - This method is more robust and secure, and the patterns are less likely to be predictable due to its random generation of the key for a given value.
  - The limitation with this type of encryption is that searching, join, group and, indexing is not possible

In an age of centralized or remote management of data, it is important that the enterprises add an abstraction layer to their data. This way, those who manage the data on a day-to-day basis, such as database administrators are not able to view or use the data. At the same time, those in the enterprise who own the data, have complete access to the data, even though they may not necessarily manage it.

Apart from being the layer of abstraction, Always Encrypted also ensures encryption of data during transit, thereby protecting it from sniffers—typically those involved in attacks such as Man in the Middle.

Configuring Always Encrypted

To set up Always Encrypted, we need to generate the following:

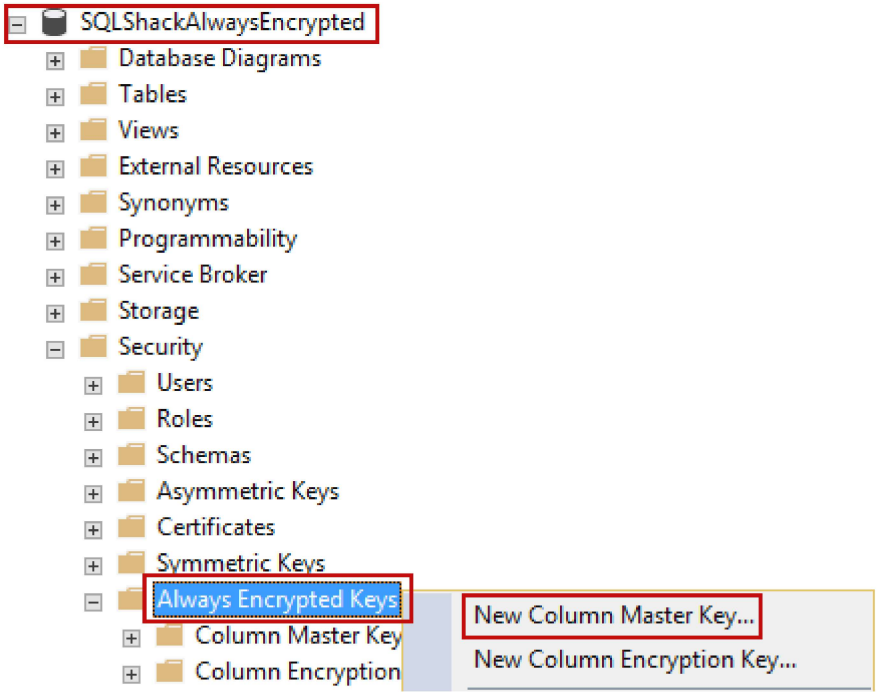
1. Key metadata
2. Encryption properties of the selected database columns, and/or encrypting the data that may already exist in columns that need to be encrypted.

However, not all of these are supported in T-SQL. Therefore, we need to use client-side tools, such as the SQL Server Management Studio or PowerShell to accomplish these tasks.

Task	SSMS	PowerShell	T-SQL
Prototyping Column Master Key and Column Encryption Key	Yes	Yes	No
Registering the Master key and Column Encryption Key metadata	Yes	Yes	Yes
Table creation with column encryption	Yes	Yes	Yes
Defining column encryption on an existing database columns	Yes	Yes	No

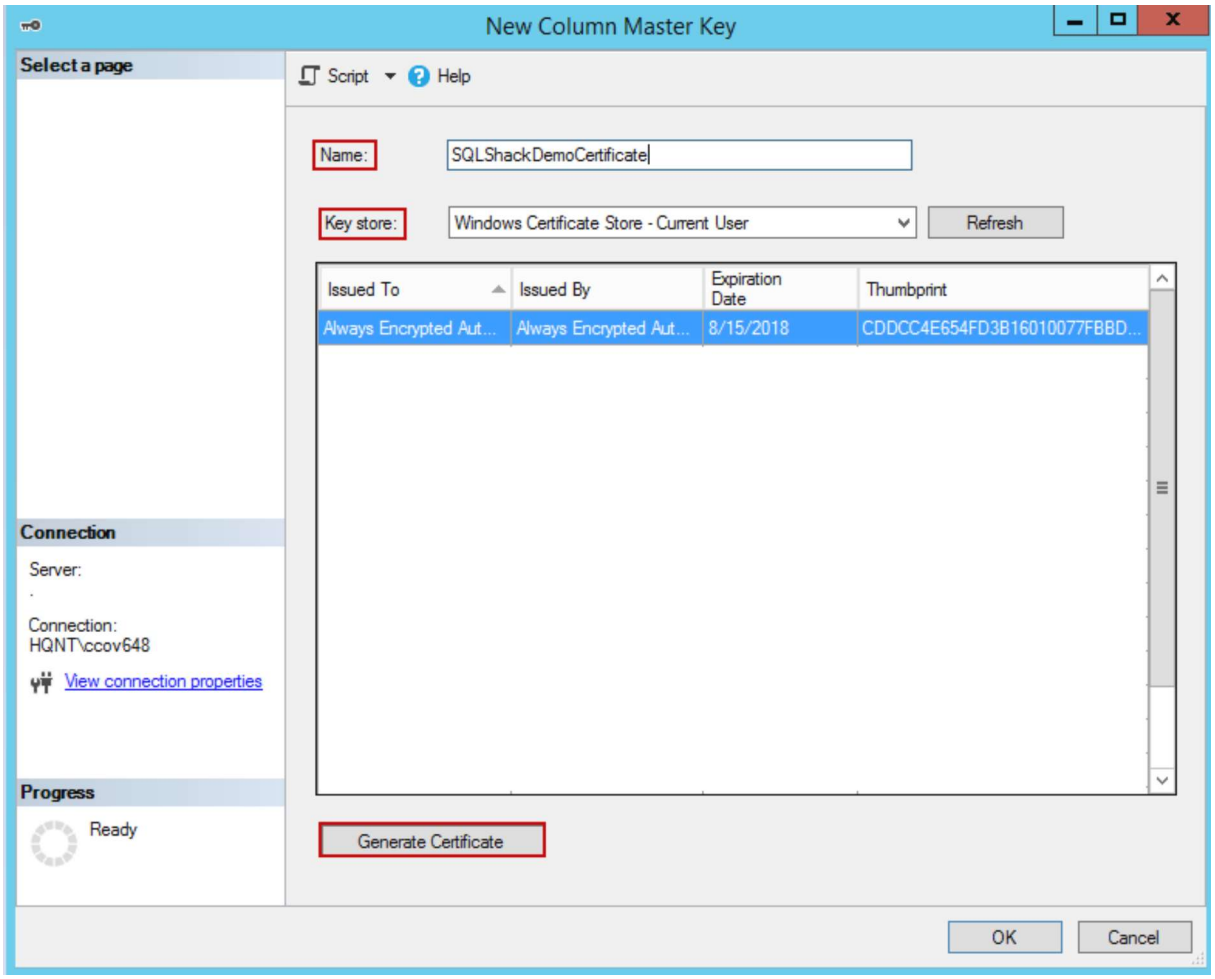
Using SSMS

- Use the **Object Explorer** to locate the database – *SQLShackAlwaysEncrypted*
- Go to the **Security** tab
- Select the **Always Encrypted Keys** option



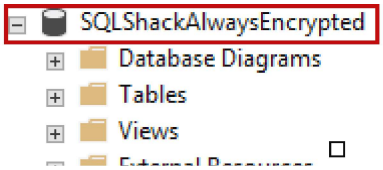
- Right-click and select **New Column Master Key...**.
- Enter the name of the Master Key *SQLShackDemoCertificate*
- Specify **Key store**, (Windows Certificates Store in this case) for the current user or local machine certificate store, or the Azure Key Vault and then select a certificate from the list. You can even one by clicking the **Generate certificate** option.
- Click **OK**

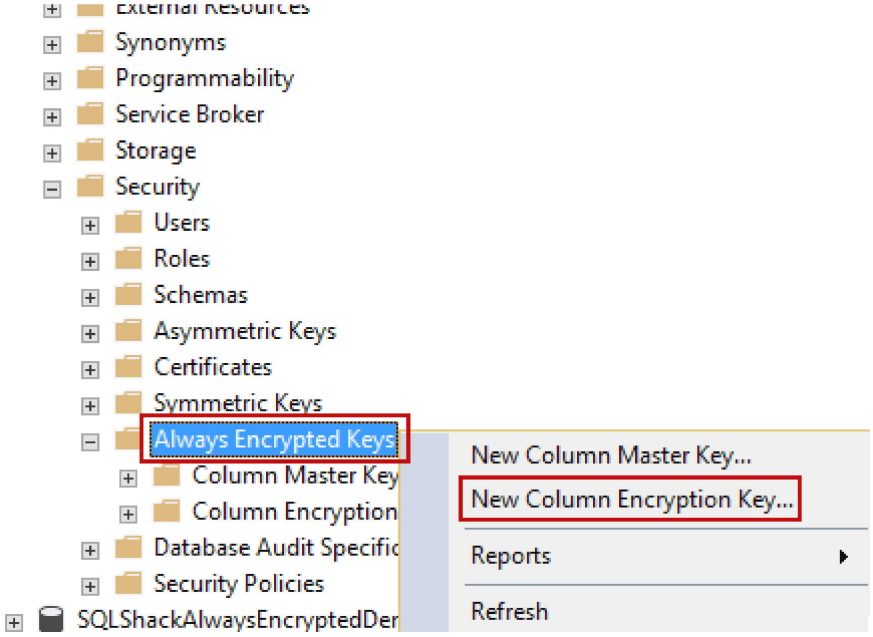
The above steps create a self-signed certificate and load it into the store.

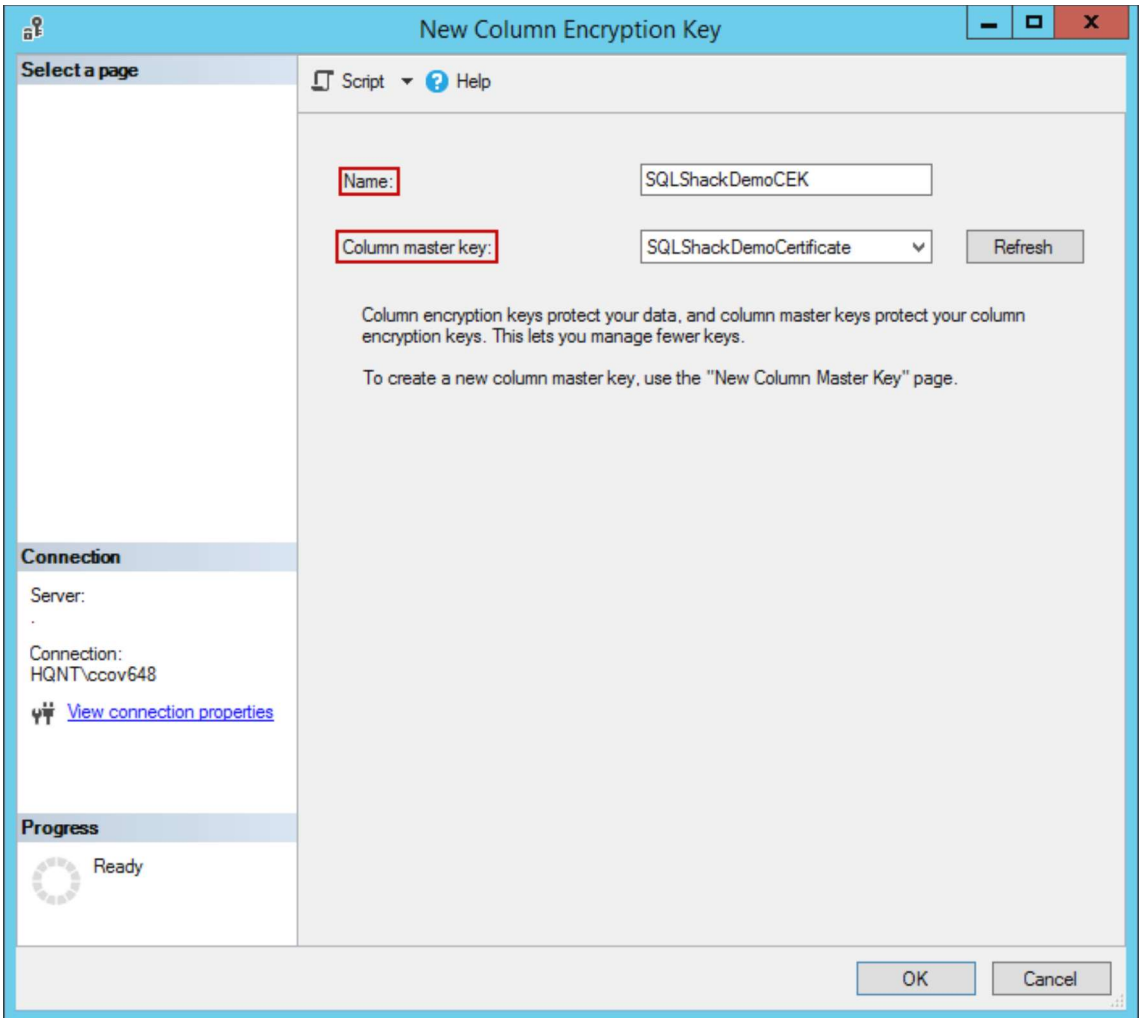


Now, we need to distribute the certificate to all the client machines by using the Export and Import Certificates method.

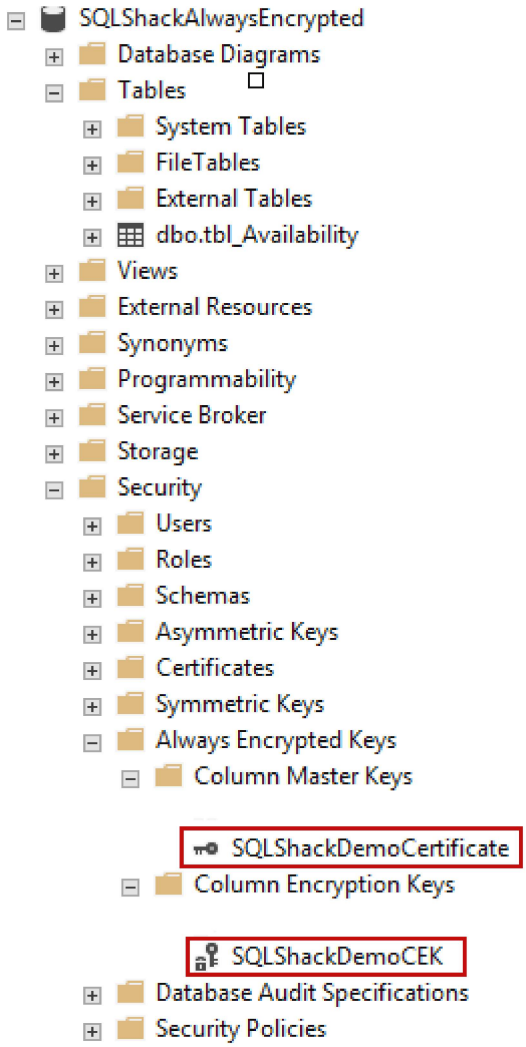
- Now, select **New Column Encryption Keys**.





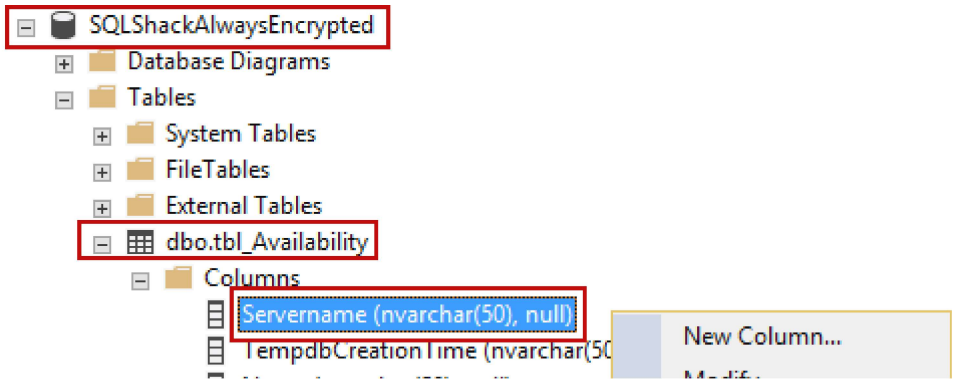


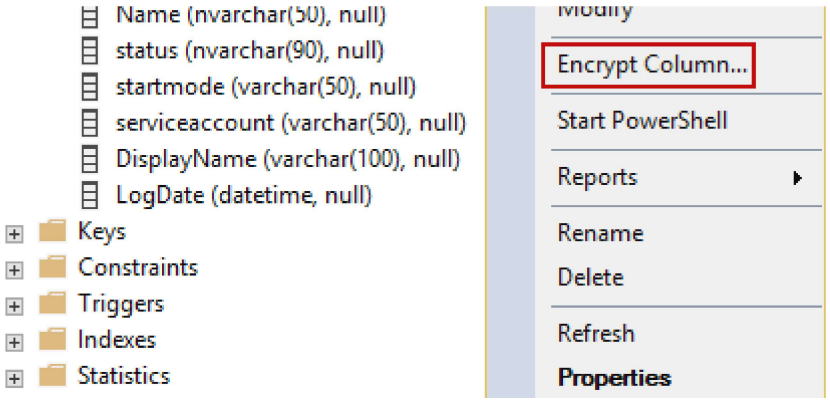
- Enter the name of the column encryption key *SQLShackDemoCEK*
- Use the drop-down and select the **Column Master Key** *SQLShackDemoCertificate*
- Click **OK**
- Verify the **Always Encrypted Keys**

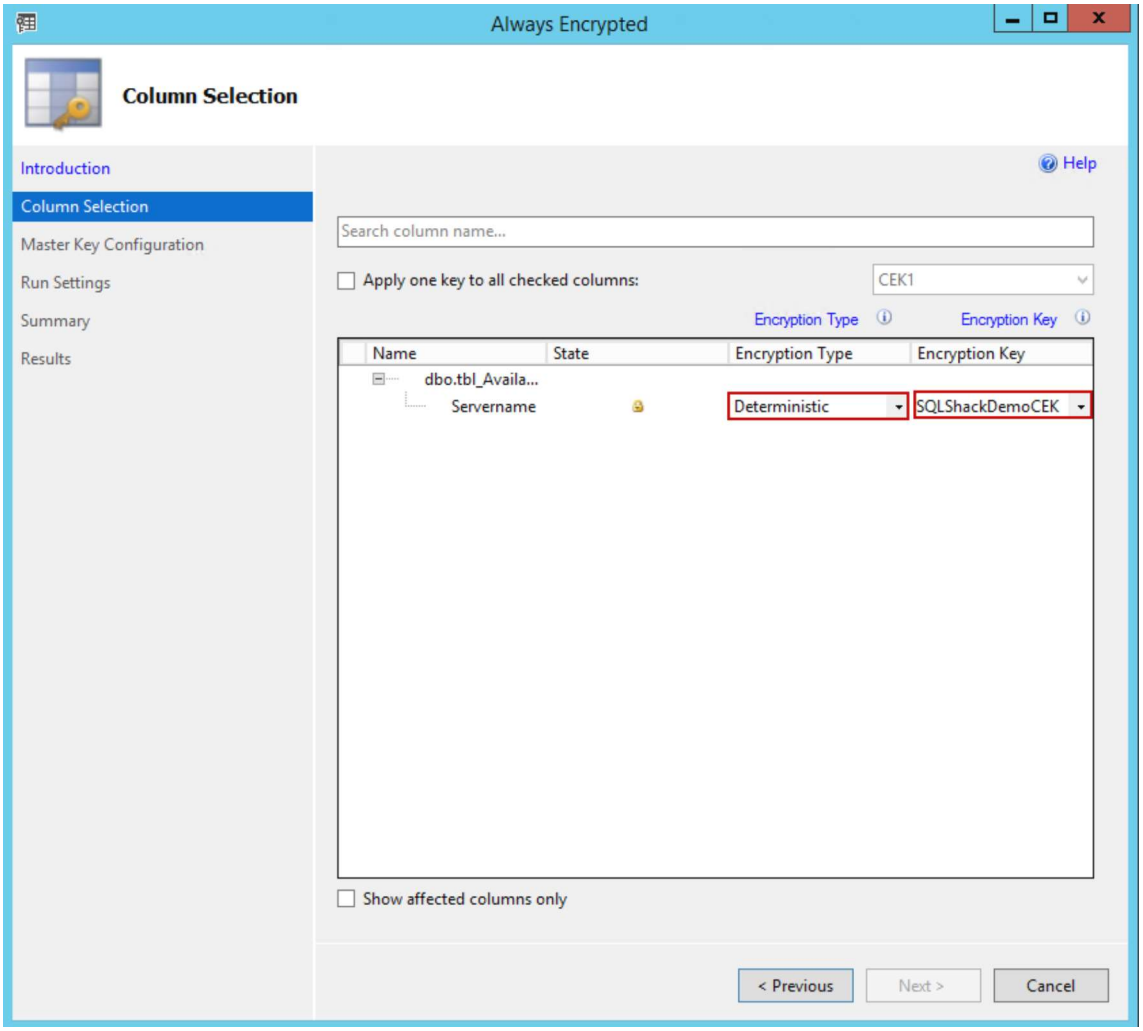


We have successfully completed the configuration. Now, it's time to apply the encryption settings to the column(s) by browsing the table and selecting the needed column(s) for encryption.

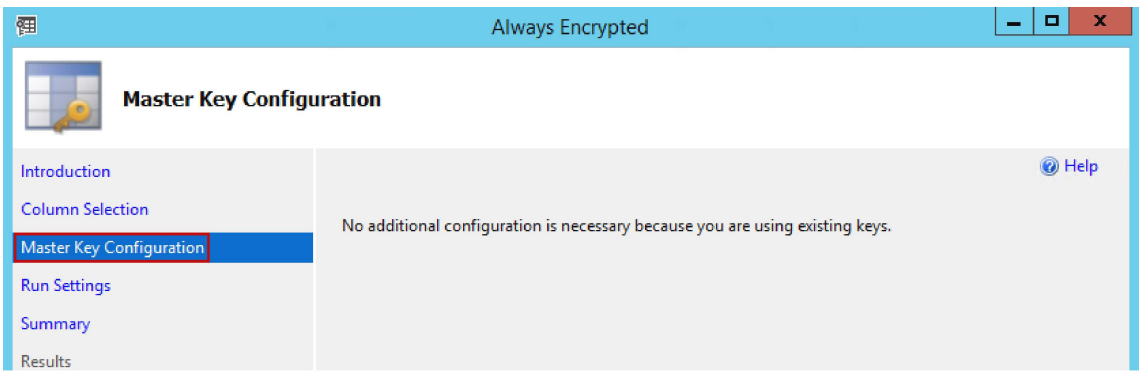
- Browse the **Columns** tab
- Right-click the column and select **Encrypt Column**
- Select the **Encryption Type**: Select either of the available options, since Always Encrypted supports two types of encryption: Randomized and Deterministic
- Use the **drop-down**, and select the **Column Encryption Key**, which is already tied with the **Column Master Key**
- Click **Next**

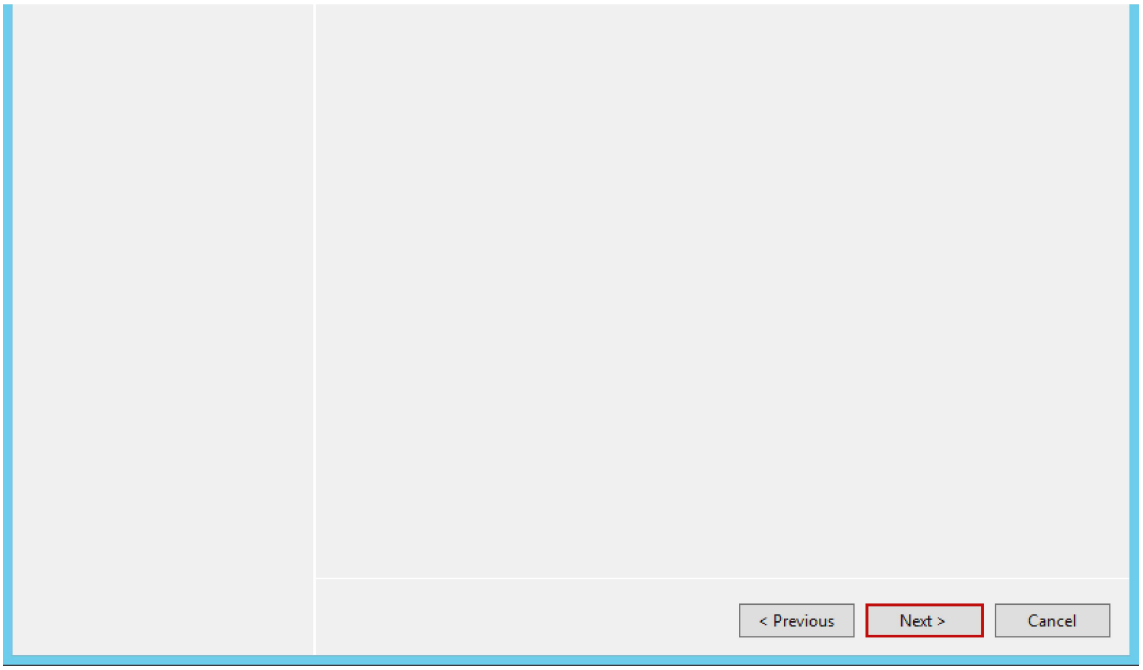



Name (nvarchar(50), null)
status (nvarchar(90), null)
startmode (varchar(50), null)
serviceaccount (varchar(50), null)
DisplayName (varchar(100), null)
LogDate (datetime, null)
Keys
Constraints
Triggers
Indexes
Statistics
Encrypt Column...
Start PowerShell
Reports
Rename
Delete
Refresh
Properties

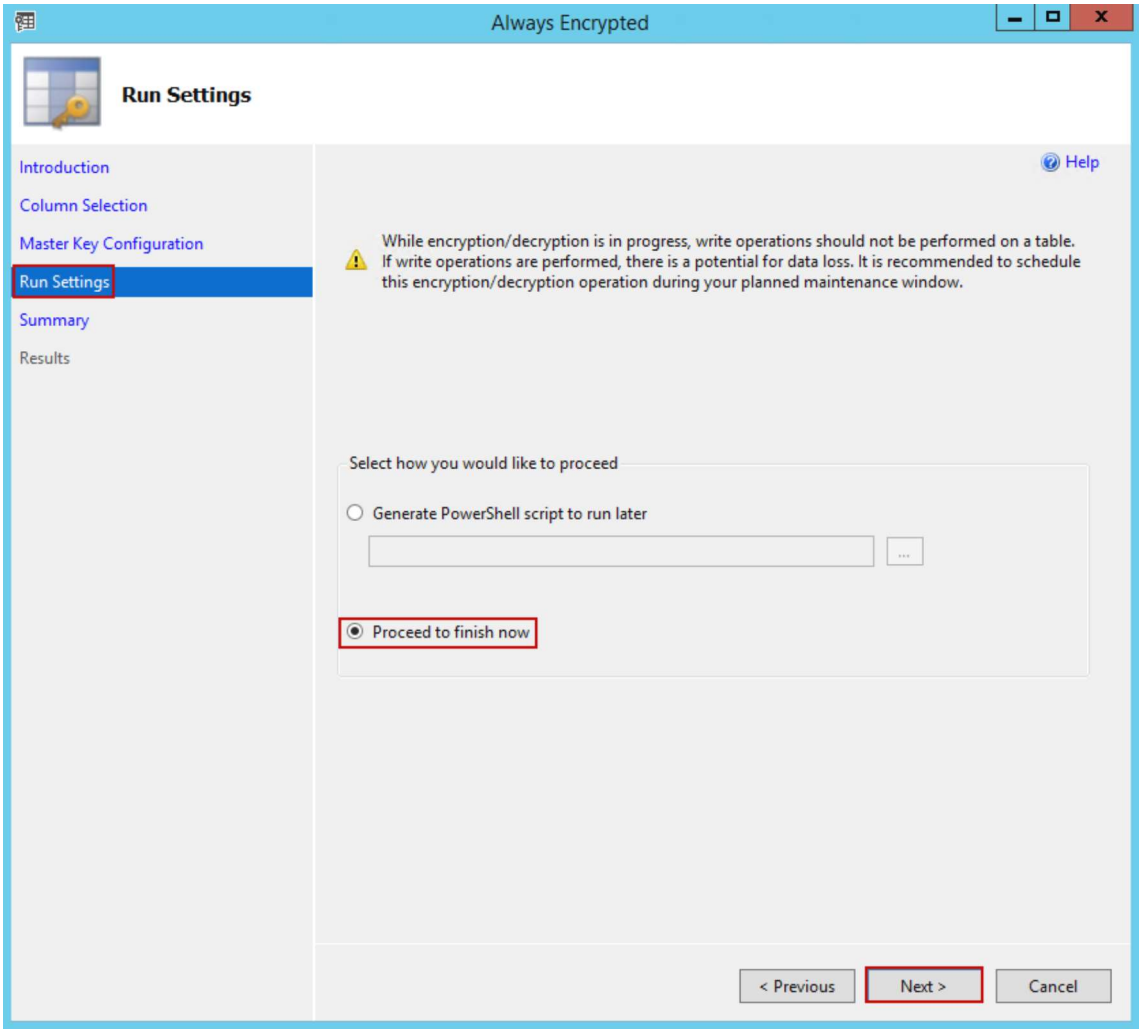

Always Encrypted
Column Selection
Introduction
Column Selection
Master Key Configuration
Run Settings
Summary
Results
Help
Search column name...
Apply one key to all checked columns:
CEK1
Encryption Type
Encryption Key
Name
State
Encryption Type
Encryption Key
dbo.tbl_Availa...
Servername
Deterministic
SQLShackDemoCEK
Show affected columns only
< Previous
Next >
Cancel

- Click **Next**

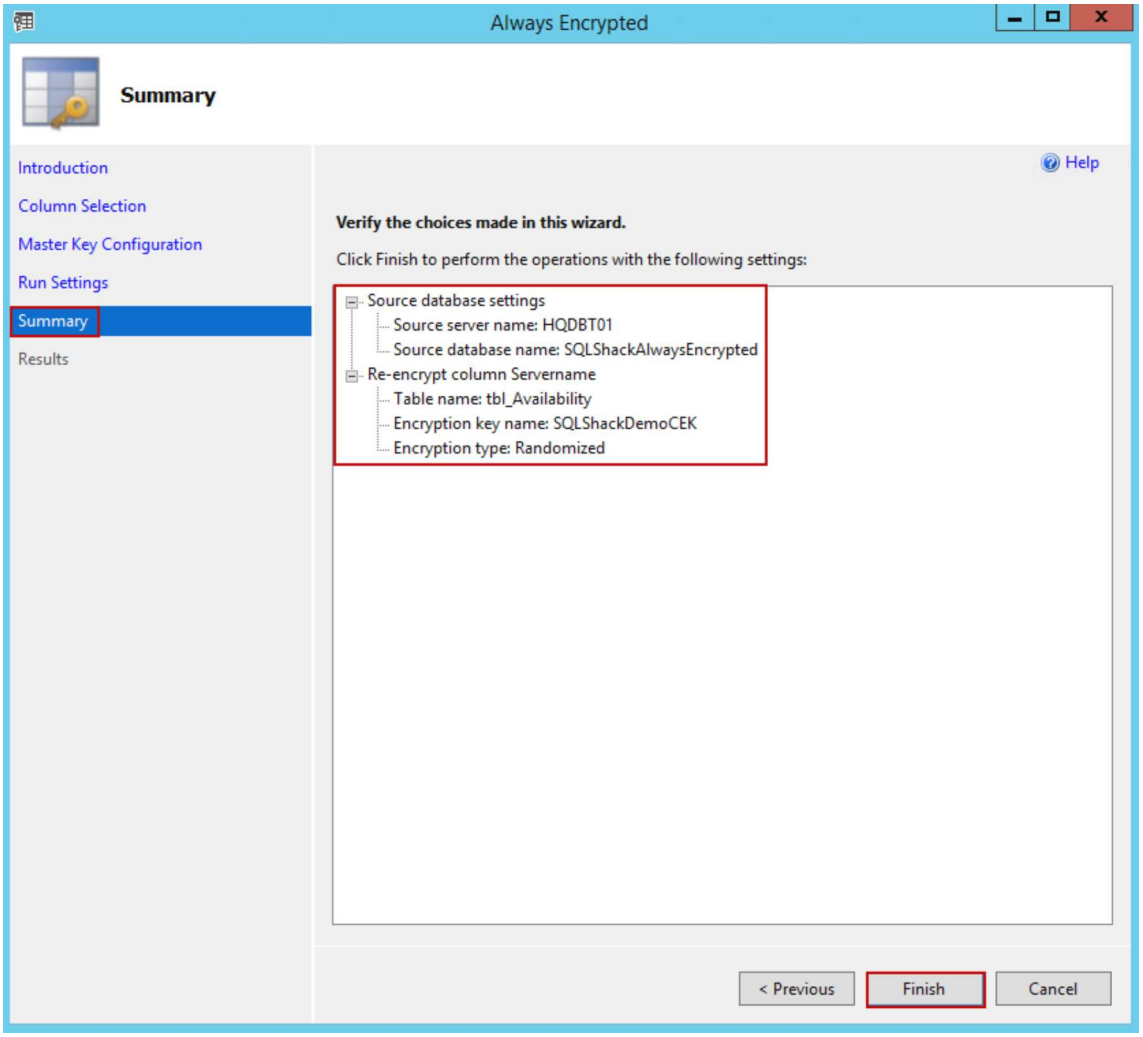

Always Encrypted
Master Key Configuration
Introduction
Column Selection
Master Key Configuration
Run Settings
Summary
Results
Help
No additional configuration is necessary because you are using existing keys.



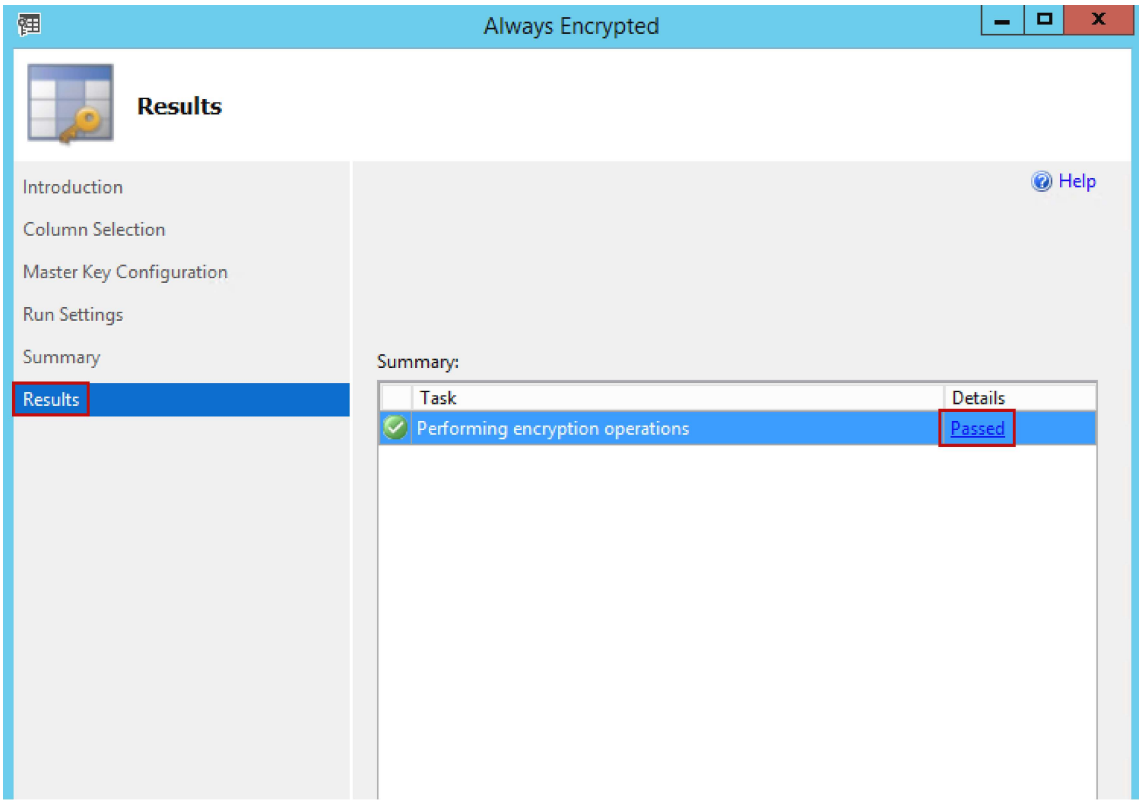
- Click default **Proceed to finish** radio button

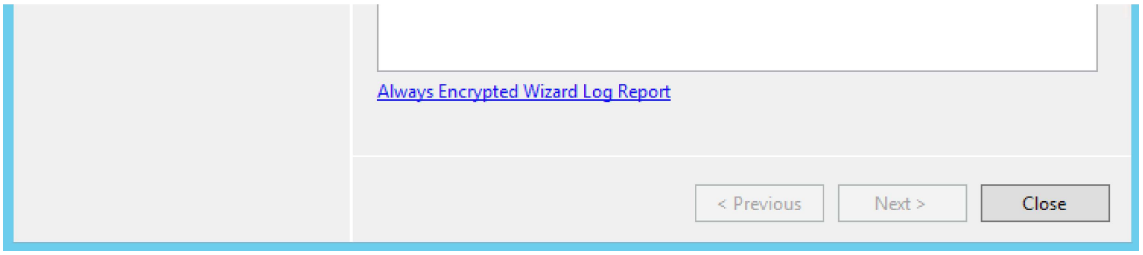


- Verify the summary of settings and click **Finish**

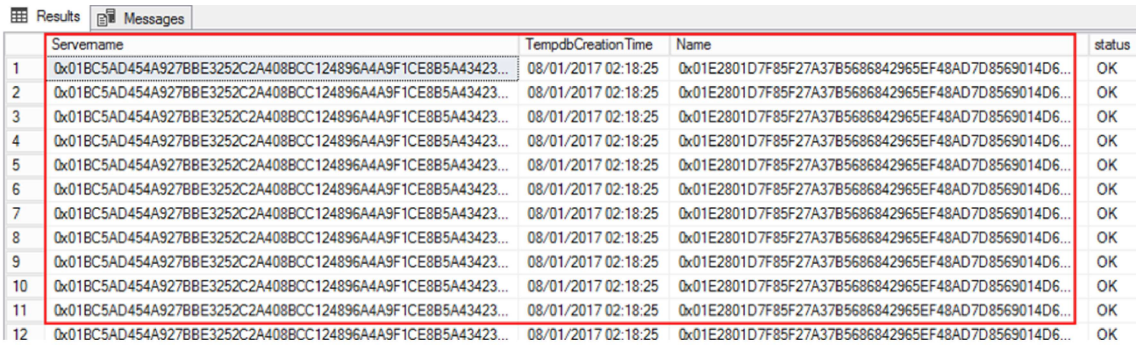


- Validate the results





- Query the table and view the encrypted column. In the below screenshot, the **Servername** column is encrypted.

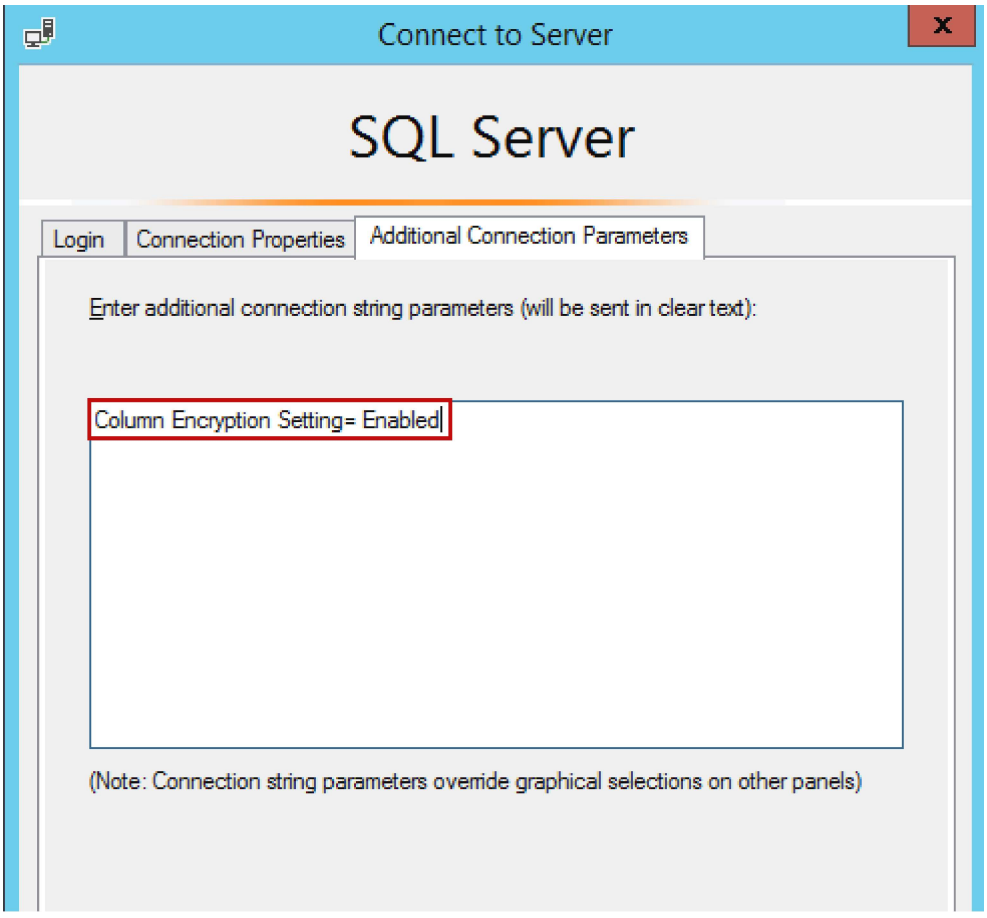


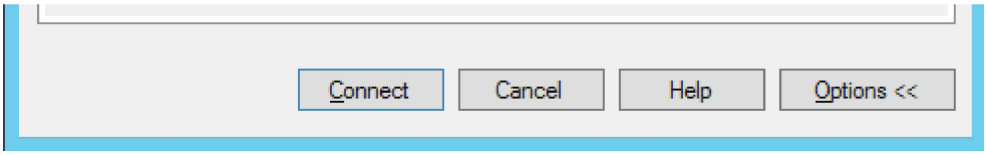
Results Messages

	Servername	TempdbCreationTime	Name	status
1	0x01BC5AD454A927BBE3252C2A408BCC124896A4A9F1CE8B5A43423...	08/01/2017 02:18:25	0x01E2801D7F85F27A37B5686842965EF48AD7D8569014D6...	OK
2	0x01BC5AD454A927BBE3252C2A408BCC124896A4A9F1CE8B5A43423...	08/01/2017 02:18:25	0x01E2801D7F85F27A37B5686842965EF48AD7D8569014D6...	OK
3	0x01BC5AD454A927BBE3252C2A408BCC124896A4A9F1CE8B5A43423...	08/01/2017 02:18:25	0x01E2801D7F85F27A37B5686842965EF48AD7D8569014D6...	OK
4	0x01BC5AD454A927BBE3252C2A408BCC124896A4A9F1CE8B5A43423...	08/01/2017 02:18:25	0x01E2801D7F85F27A37B5686842965EF48AD7D8569014D6...	OK
5	0x01BC5AD454A927BBE3252C2A408BCC124896A4A9F1CE8B5A43423...	08/01/2017 02:18:25	0x01E2801D7F85F27A37B5686842965EF48AD7D8569014D6...	OK
6	0x01BC5AD454A927BBE3252C2A408BCC124896A4A9F1CE8B5A43423...	08/01/2017 02:18:25	0x01E2801D7F85F27A37B5686842965EF48AD7D8569014D6...	OK
7	0x01BC5AD454A927BBE3252C2A408BCC124896A4A9F1CE8B5A43423...	08/01/2017 02:18:25	0x01E2801D7F85F27A37B5686842965EF48AD7D8569014D6...	OK
8	0x01BC5AD454A927BBE3252C2A408BCC124896A4A9F1CE8B5A43423...	08/01/2017 02:18:25	0x01E2801D7F85F27A37B5686842965EF48AD7D8569014D6...	OK
9	0x01BC5AD454A927BBE3252C2A408BCC124896A4A9F1CE8B5A43423...	08/01/2017 02:18:25	0x01E2801D7F85F27A37B5686842965EF48AD7D8569014D6...	OK
10	0x01BC5AD454A927BBE3252C2A408BCC124896A4A9F1CE8B5A43423...	08/01/2017 02:18:25	0x01E2801D7F85F27A37B5686842965EF48AD7D8569014D6...	OK
11	0x01BC5AD454A927BBE3252C2A408BCC124896A4A9F1CE8B5A43423...	08/01/2017 02:18:25	0x01E2801D7F85F27A37B5686842965EF48AD7D8569014D6...	OK
12	0x01BC5AD454A927BBE3252C2A408BCC124896A4A9F1CE8B5A43423...	08/01/2017 02:18:25	0x01E2801D7F85F27A37B5686842965EF48AD7D8569014D6...	OK

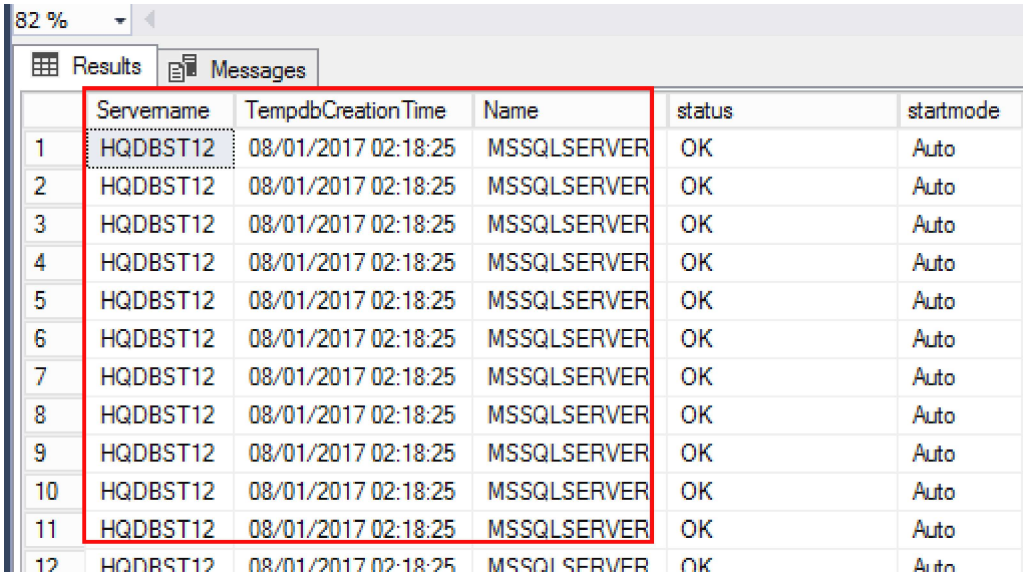
In order to decrypt the column, the following settings should be enabled in the SSMS client

- First, add **Column Encryption Setting = Enabled** in the **Additional Connection Parameters** in the SSMS **Connect to Server** window**.**





- Now, query the table for the encrypted values



	Servername	TempdbCreationTime	Name	status	startmode
1	HQDBST12	08/01/2017 02:18:25	MSSQLSERVER	OK	Auto
2	HQDBST12	08/01/2017 02:18:25	MSSQLSERVER	OK	Auto
3	HQDBST12	08/01/2017 02:18:25	MSSQLSERVER	OK	Auto
4	HQDBST12	08/01/2017 02:18:25	MSSQLSERVER	OK	Auto
5	HQDBST12	08/01/2017 02:18:25	MSSQLSERVER	OK	Auto
6	HQDBST12	08/01/2017 02:18:25	MSSQLSERVER	OK	Auto
7	HQDBST12	08/01/2017 02:18:25	MSSQLSERVER	OK	Auto
8	HQDBST12	08/01/2017 02:18:25	MSSQLSERVER	OK	Auto
9	HQDBST12	08/01/2017 02:18:25	MSSQLSERVER	OK	Auto
10	HQDBST12	08/01/2017 02:18:25	MSSQLSERVER	OK	Auto
11	HQDBST12	08/01/2017 02:18:25	MSSQLSERVER	OK	Auto
12	HQDBST12	08/01/2017 02:18:25	MSSQLSERVER	OK	Auto

- And voila!

Using PowerShell

Let us now go through the step-by-step procedure of configuring Always Encrypted using PowerShell.

On Windows 2016, the creation of **New-SelfSignedCertificate** is pretty straight forward, and it comes with a long list of parameters. The example which I'm walking through is done on Windows 2012. This has a significant limitation on the type of certificate that can be created using PowerShell and the APIs. The **New-SelfSignedCertificateEx** is an enhanced version of Windows 2012 New-SelfSignedCertificate cmdlet.

Step 1: Certificate Management

The first step is to create a self-signed certificate with all the necessary information related to loading it into the certificate store of the current user context.

- Create certificate with **KeyUsage** as *DataEncipherment* and a **friendlyname**
- The list of other parameters used in the self-signed certificate creation process is explained below

After downloading, create the function *New-SelfsignedCertificateEx* and call the function with the available parameters as shown below

available parameters as shown below

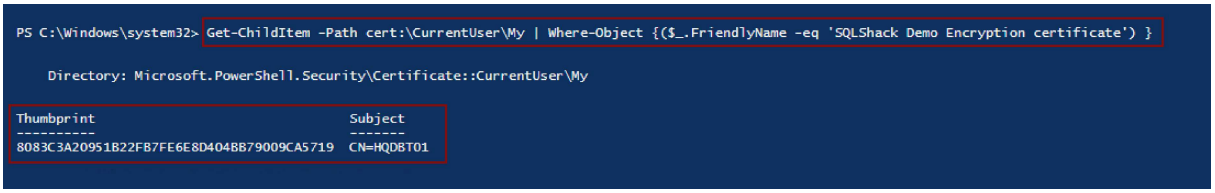
```
$certificate=New-SelfsignedCertificateEx `
    -Subject "CN=${ENV:ComputerName}" `
    -EKU 'Document Encryption' `
    -KeyUsage 'KeyEncipherment, DataEncipherment' `
    -FriendlyName 'SQLShack Demo Encryption certificate' `
    -Exportable `
    -StoreLocation 'CurrentUser' `
    -KeyLength 2048 `
    -ProviderName 'Microsoft Enhanced Cryptographic Provider v1.0' `
    -AlgorithmName 'RSA' `
    -SignatureAlgorithm 'SHA256'
```

1. The KeyUsage — this parameter defines the purpose of the public key contained in the certificate. It's a way of providing restrictions on the operations that can be performed by the public key. With DataEncipherment, the public key is used to encrypt user data, apart from the cryptographic keys.
2. SignatureAlgorithm —the default 'SHA1' algorithm is used.
3. FriendlyName — specifies a friendly name for the certificate.
4. StoreLocation — specifies the location to store self-signed certificate. Possible values are 'CurrentUser' and 'LocalMachine'. 'CurrentUser' store is intended for user certificates; computer (as well as CA) certificates are usually stored in the 'LocalMachine' store.

Let's proceed further.

- Locate the newly-created certificate *SQLShack Demo Encryption certificate*

```
Get-ChildItem -Path cert:\CurrentUser\My | Where-Object {($_.FriendlyName -eq 'SQLShack Demo Encryption
certificate') }
```



- Export the certificate using the **Export-Certificate** cmdlet

```
PS C:\Windows\system32> $SQLShackCertificate = Get-ChildItem -Path cert:\CurrentUser\My |
Where-Object {($_.FriendlyName -eq 'SQLShack Demo Encryption certificate') }
PS C:\Windows\system32> $SQLShackCertificate | Export-Certificate -FilePath
"F:\PowerSQL\SQLShackAEPublic.cer" -Force
```

```
PS C:\Windows\system32> $SQLShackCertificate = Get-ChildItem -Path cert:\CurrentUser\My | Where-Object {($_.FriendlyName -eq 'SQLShack Demo Encryption certificate') }
PS C:\Windows\system32> $SQLShackCertificate | Export-Certificate -FilePath "F:\PowerSQL\SQLShackAEPublic.cer" -Force


    Directory: F:\PowerSQL

Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----        8/24/2017   2:27 PM            761 SQLShackAEPublic.cer
```

We can copy the certificates to all intended client machines by manually copying the files. To install the certificate, right-click and select *install the certificate* and follow the instructions. Alternatively, we can also use the **Import-certificate** cmdlet to import the certificates.

Step 2: Import SQL Server module

The **SqlServer** module is an external package. Hence this has to be installed as a separate package. You can download and install SqlServer in very few simple steps.

Import-Module SqlServer

```
PS C:\Windows\system32> Import-Module "SqlServer"
```

Step 3: Define the Connection String

This step is to prepare the SQL Server connection string and prepare the database to be Always Encrypted. Once the SQL Server module is loaded, it's very simple and straight forward to define the connection string and use it.

```
$sqlConnectionString = "Data Source=hqdbt01;Initial Catalog=SQLShackDemo;Integrate
d
Security=True;MultipleActiveResultSets=False;Connect
Timeout=30;Encrypt=False;TrustServerCertificate=True;Packet Size=4096;Application
Name=`"Microsoft SQL Server Management Studio`""
$database = Get-SqlDatabase -ConnectionString $sqlConnectionString
```

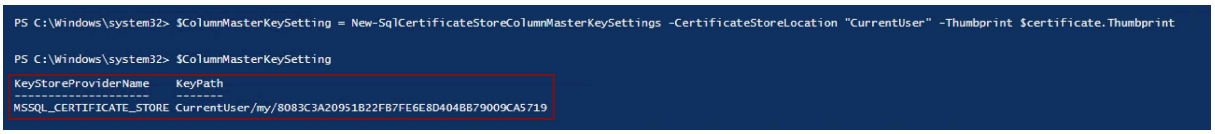
```
PS C:\Windows\system32> $sqlConnectionString = "Data Source=hqdbt01;Initial Catalog=SQLShackDemo;Integrated Security=True;MultipleActiveResultSets=False;Connect Timeout=30;Encrypt=False;TrustServerCertificate=True;
$database = Get-SqlDatabase -ConnectionString $sqlConnectionString

PS C:\Windows\system32> $database
Name                 Status           Size     Space Available  Recovery Model  Compat. Level  Owner                Collation                 Containment Type
SQLShackDemo         Normal      404.00 MB   67.09 MB Full            130 HQNT\CCOV648     SQL_Latin1_General_CP1_CI_AS  None
```

Step 4: Create Master Column Encryption Key (CMK)

Create a ColumnMasterKeySettings object using the *New-SqlCertificateStoreColumnMasterKeySettings*

cmdlet. This loads the certificate metadata into a variable called $ColumnMasterKeySetting. This setting variable is referred while creating the ColumnMasterKey.

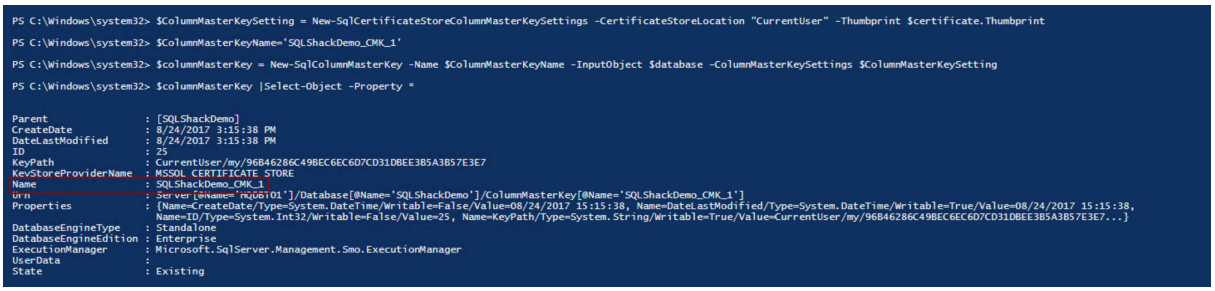
```
$ColumnMasterKeySetting = New-SqlCertificateStoreColumnMasterKeySettings -
CertificateStoreLocation "CurrentUser" -Thumbprint $certificate.Thumbprint
```



To create the ColumnMasterKey (CMK), the cmdlet New-SqlColumnMasterKey is used, which requires the following references as its parameters

- Name of the CMK
- Database
- CMK settings

```
$ColumnMasterKeyName='SQLShackDemo_CMK_1'

$columnMasterKey = New-SqlColumnMasterKey -Name "SQLShackCMK" -InputObject $databa
se -
ColumnMasterKeySettings $ColumnMasterKeySetting
```



That's it; we have our Column Master Key now.

Step 5: Configure the Column Encryption Key (CEK)

Let's now proceed to create the Column Encryption Keys. The .NET driver enables the use of Column Encryption Keys to encrypt and decrypt the data during data exchange between the client and the SQL Server. The driver provides the extra layer of protection in order to secure the data during interchange. The *New-SqlColumnEncryptionKey* cmdlet is being used to create the Column Encryption Key. This requires three input parameters:

- CEK (Column Encryption Key) name
- Database deference
- CMK Name

- CMK Name

```
$columnEncryptionKeyName = "SQLShackDemo_CEK_1"
$ColumnEncryptionKey=New-SqlColumnEncryptionKey -Name $columnEncryptionKeyName  -
InputObject $database -ColumnMasterKey $ColumnMasterKeyName
```

```
PS C:\Windows\system32> $columnEncryptionKeyName = "SQLShack_CEK_1"
PS C:\Windows\system32> $ColumnEncryptionKey=New-SqlColumnEncryptionKey -Name $columnEncryptionKeyName  -InputObject $database -ColumnMasterKey $ColumnMasterKeyName
```

```
$ColumnEncryptionKey |Select-Object -Property *
```

```
PS C:\Windows\system32> $ColumnEncryptionKey |Select-Object -Property *

Parent                  : [SQLShackDemo]
CreateDate              : 8/24/2017 3:23:54 PM
DateLastModified        : 8/24/2017 3:23:54 PM
ID                      : 16
Name                    : SQLShack_CEK_1
ColumnEncryptionKeyValues : {SQLShack_CEK_1}
Urn                     : Server[@Name='HQDBT01']/Database[@Name='SQLShackDemo']/ColumnEncryptionKey[@Name='SQLShack_CEK_1']
Properties              : {Name=CreateDate/Type=System.DateTime/Writable=False/Value=08/24/2017 15:23:54, Name=DateLastModified/Type=System.DateTime/Writable=True/Value=08/24/2017 15:23:54, |
                          Name=ID/Type=System.Int32/Writable=False/Value=16}
DatabaseEngineType      : Standalone
DatabaseEngineEdition   : Enterprise
ExecutionManager        : Microsoft.SqlServer.Management.Smo.ExecutionManager
UserData                :
State                   : Existing
```

Step 6: Migrate the schema

It's time to integrate the columns with the Always Encrypted feature now. In the following steps, the column, *name*, and the database *databases* are encrypted using SQLShack_CEK_1 Column Encryption Key.

```
# Change encryption schema
$changes = @()
# Add changes for table [dbo].[databases]
$changes += New-SqlColumnEncryptionSettings -ColumnName dbo.databases.name -
EncryptionType Deterministic -EncryptionKey "SQLShack_CEK_1"
Set-SqlColumnEncryption -ColumnEncryptionSettings $changes -InputObject $database
```

Step 7: Encryption validation

We're all set for the testing and validation of the data. Let's try to access the data by setting two connection strings, one with Column Encryption setting and the other without the Encryption setting.

We can see that the Name column is encrypted and the contents look like a series of numbers as shown below. We can decrypt the column by setting Column Encryption Setting to Enabled. Now, the data in the Name column is readable.

```
$serverName = "hqdbt01"
```

```
$databaseName = "SQLShackDemo"
$strConn = "Server = " + $serverName + "; Database = " + $databaseName + "; Integr
ated
Security = True"
Invoke-Sqlcmd -Query "SELECT TOP(10) * FROM databases" -ConnectionString $strConn|
format-table -AutoSize
```

Did that succeed? Let's now try connecting, using a connection string set to use the Column Encryption Setting.

```
$strConn = $strConn + "; Column Encryption Setting = Enabled"
Invoke-Sqlcmd -Query "SELECT TOP(10) * FROM databases" -ConnectionString $strConn|
format-table -AutoSize
```

Using T-SQL

This section talks about using T-SQL to create Column Master Key and Column Encryption Key along with creating encrypted columns in a table. The following are the three tables that are very important to get the required key information to create Master and Column encryption keys.

- sys.column_master_keys
- sys.column_encryption_keys
- sys.column_encryption_key_values

Step 1: Create Column Master Key

To get the provider name and the key path details, query the system view `sys.column_master_keys`

```
SELECT
    Name,
    key_store_provider_name KeyStore,
    key_path KeyPath
FROM sys.column_master_keys
```

Use the **create column master key** DDL to define the *SQLShackAECMK* master key

```
CREATE COLUMN MASTER KEY SQLShackAECMK
WITH (
```

```
    KEY_STORE_PROVIDER_NAME = 'MSSQL_CERTIFICATE_STORE',
    KEY_PATH = 'Current User/my/96B46286C49BEC6EC6D7CD31DBEE3B5A3B57E3E7'
  );
```

Step 2: Create Column Encryption Key

To get encrypted_value and algorithm details, query the following system views `sys.column_encryption_key_values` and `sys.column_encryption_keys`

```
SELECT
    NAME,
    ENCRYPTED_VALUE ,
    ENCRYPTION_ALGORITHM_NAME ALGORITHM
FROM
    sys.column_encryption_key_values CEKV
    inner join
    sys.column_encryption_keys CEK
    ON
        CEKV.column_encryption_key_id=CEK.column_encryption_key_id
WHERE
    NAME='SQLShack_CEK_1'
```

Use the **create column encryption key** DDL to create the *SQLShackAECEK* CEK key

```
CREATE COLUMN ENCRYPTION KEY SQLShackAECEK
WITH VALUES
(
    COLUMN_MASTER_KEY = SQLShackAECMK,
    ALGORITHM = 'RSA_OAEP',
    ENCRYPTED_VALUE =
0x016E000001630075007200720065006E00740075007300650072002F006D0079002F003900360062
0034003
600320038003600630034003900620065006300360065006300360064003700630064003300310064
0620065
006500330062003500610033006200350037006500330065003700820F6F9357AE51A486DFBA98D83B3
87CDE04
5DE8DD9630C1F55ED306BA559CA2BF3CA12C4926C30A0BF4F7C5A51D2F4ACA160B48577DFC21D81D26
82F0C23
A1F0C1B95BADF1C587077F1312D7C511A690C9E74B80C0725C1BBBC84C7557892B8F4AF575774A6291
B19A313
F5E1975AC4087162F3DECF5BD68F34A553DC39B42A6FC943219687A632FC308F19CDAAA9D0ED137C3C
F64827F
713EBA58A33FA20C11FA8D917D3ED6572EFF6389477BD170EE9B9889D31185B0BD1BC98FFC88550EC8
54D8ECD
DD4D7F4BC30CD4482A5DDD90354986BC00C3D576A57A095226D9508863B71866688BE4B16F8FAA8BF3
EC4E2CE
09C69A8317B8B32D887F85EB78F22892748CABEC8FA277412CF11C130E00E921E64F8D84D10BDE770E
91B2A4D
0CE6FBB5BFF6196B9A6020E9F750DC7837BCA5E3E6C092169C00306B64B3FA3FF70AB409380144A5A3
D711C47
C0C0D80484B4237A25F03406B1A42A43B68F56DED431FA53D17F7F1B799A1BFBFBEE078D3EAC087E12
7D86778
9BEE595AFAA8C249861AAD9316DEAF4C60552D69ED8E42E4D5E942A1EA62E156F3EF6652FFB0A047DF
15FBD43
```

```
A44E7CFFB2F4BBFEC71FB8F34C59BBFF1ED05E834332F93C279233E3044DBB3DEB004DB8D67AF75B04
4CCEB7D
192E6471090ACE9067E8BBCEFFED40962555853E6EE0D90A37AD874FF1E42F621B39C5F22AA84BAE49
DFD7B4E
F40EDE2A5E
);
```

Step 3: Create Encrypted Columns table

Use the **create table** DDL with a few additional configurations in the column definition

```
CREATE TABLE tbl_CustomerSQLShackDemoAE (
    custName nvarchar(30)
        COLLATE  Latin1_General_BIN2 ENCRYPTED WITH (
        ENCRYPTION_TYPE = RANDOMIZED,
        ALGORITHM = 'AEAD_AES_256_CBC_HMAC_SHA_256',
        COLUMN_ENCRYPTION_KEY = SQLShackAECEK),
    custSSN varchar(11)
        COLLATE  Latin1_General_BIN2 ENCRYPTED WITH (
        ENCRYPTION_TYPE = DETERMINISTIC ,
        ALGORITHM = 'AEAD_AES_256_CBC_HMAC_SHA_256',
        COLUMN_ENCRYPTION_KEY = SQLShackAECEK),
    custBirthDate [date]
        ENCRYPTED WITH (ENCRYPTION_TYPE = RANDOMIZED,
        ALGORITHM = 'AEAD_AES_256_CBC_HMAC_SHA_256',
    COLUMN_ENCRYPTION_KEY = SQLShackAECEK) NOT NULL,
    CustAge int NULL,

);
```

That's all.

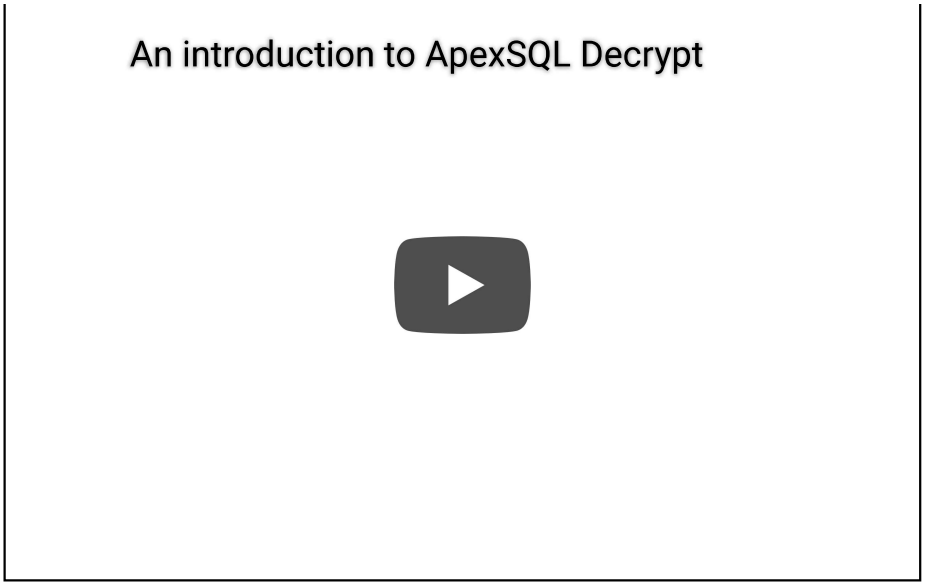
Conclusion

In this article, we saw various ways to configure and enable the Always Encryption feature. Since data to the database flows from various sources, this feature gives an added security to the data and safeguards the same from various potential risks.

However, before proceeding with this, I recommend that all of the implications are fully understood, and the feature details are known.

See more

Be sure to check out ApexSQL Decrypt, a free add-in that decrypts SQL Server database objects directly from SSMS



Prashanth Jayaram

I'm a Database technologist having 11+ years of rich, hands-on experience on Database technologies. I am Microsoft Certified Professional and backed with a Degree in Master of Computer Application.

My specialty lies in designing & implementing High availability solutions and cross-platform DB Migration. The technologies currently working on are SQL Server, PowerShell, Oracle and MongoDB.

View all posts by Prashanth Jayaram

Related Posts:

1. **Certificate Management in SQL Server 2019**
2. **An overview of the column level SQL Server encryption**
3. **Understanding Database Backup Encryption in SQL Server**
4. **How to add a TDE encrypted user database to an Always On Availability Group**
5. **How to configure SQL Server mirroring on a TDE encrypted database**

PowerShell, Security, SQL Server 2016, SQL Server Management Studio (SSMS), T-SQL

168 Views

ALSO ON SQL SHACK

5 months ago • 1 comment
Lever T-SQL for Pinpoint Control of …

4 months ago • 1 comment
Starting your Journey with Azure Data

4 mon
Mig BI bet

Comments Community Privacy Policy 1 Login

Recommend 10 Tweet Share Sort by Best

Join the discussion…

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name

aijaz • 3 years ago

Hi,
I need use PowerShell to initiate a connection using a cert and mimic an application connecting to the encrypted database. pls help

7 • Reply • Share ›

Prashanth Jayaram aijaz • 3 years ago

Hi Aijaz,
Thanks for the comment. This article in-houses script to configure AE using PowerShell, T-SQL and SSMS. You can refer the PowerShell section to get more information. Let me know in case of any further information required.

Best Regards,
Prashanth

6 • Reply • Share ›

Haider Ali • 3 years ago

Great Article. Great work Prashanth

6 • Reply • Share ›

Prashanth Jayaram ➚ Haider Ali • 3 years ago

Thank you, Haider Ali! Thanks for reading my space.

6 • Reply • Share ›

abel winston veloria ➚ Prashanth Jayaram • 3 years ago

Hi Prashanth , you did a great explanation. Can u send to my mail the same set connecting remotely to SQL server always encrypted but using C#.Net? I'm using .Net ... Thank you in advace

6 • Reply • Share ›

Prashanth Jayaram ➚ abel winston veloria • 3 years ago

Hello,
Thanks for the comment. In the connection string enable the column encryption setting parameter.
For example,
string connStr = "Data Source=HQDBSP07; Initial Catalog=SQLShack; Integrated Security=true; Column Encryption Setting=enabled";

For more information, please refer the below link

https://docs.microsoft.com/...

Best Regards,
Prashanth

6 • Reply • Share ›

Saurabh Jain • 2 years ago

Hi Prashanth, while executing the command
Set-SqlColumnEncryption -InputObject $smoDatabase -ColumnEncryptionSettings $ces
I am getting this error
Set-SqlColumnEncryption : An Azure authentication context

has not been properly established or it has expired.
At line:1 char:1
+ Set-SqlColumnEncryption -InputObject $smoDatabase -ColumnEncryptionSe ...
+
~~~~~~
+ CategoryInfo : InvalidOperation: (:) [Set-SqlColumnEncryption], AzureAuthContextNotSetException
+ FullyQualifiedErrorId : EncryptionError,Microsoft.SqlServer.Management.PowerShe

Instead when I am configuring the AlwaysEncrypted from SSMS then it successfully done with the following warning messages

Jun 6 2018 17:33:08 [Warning] DacFxMigration: Message:sp_refresh_parameter_encryption failed to refresh module '[dbo].[GetActiveTagList]'. Error: Encryption scheme mismatch for columns/variables 'Name'. The encryption scheme for the columns/variables is (encryption_type = 'DETERMINISTIC', encryption_algorithm_name = 'AEAD_AES_256_CBC_HMAC_SHA_256', column_encryption_key_name = 'CEK2', column_encryption_key_database_name = 'ApplicationDB') and the expression near line '20' expects it to be (encryption_type = 'PLAINTEXT') (or weaker). .

Jun 6 2018 17:33:08 [Warning] DacFxMigration: Message:sp_refresh_parameter_encryption failed to refresh module '[dbo].[GetMediaListForPresentation]'. Error: The data types nvarchar(max) encrypted with (encryption_type = 'DETERMINISTIC', encryption_algorithm_name = 'AEAD_AES_256_CBC_HMAC_SHA_256', column_encryption_key_name = 'CEK2'
~~~~~~