securosys

SWISS SECURITY TECHNOLOGIES
FOR COMMUNICATIONS SYSTEMS

Securing Microsoft SQL Server

Using Primus Hardware Security Module
Application Note

Primus HSM Integration Guide for
Microsoft SQL Server 2016

Securosys SA, Förrlibuckstrasse 70
CH-8005 Zürich, Switzerland
Tel. +41 44 552 31 00 • www.securosys.ch
info@securosys.ch

Microsoft Partner

securosys

Table of Contents

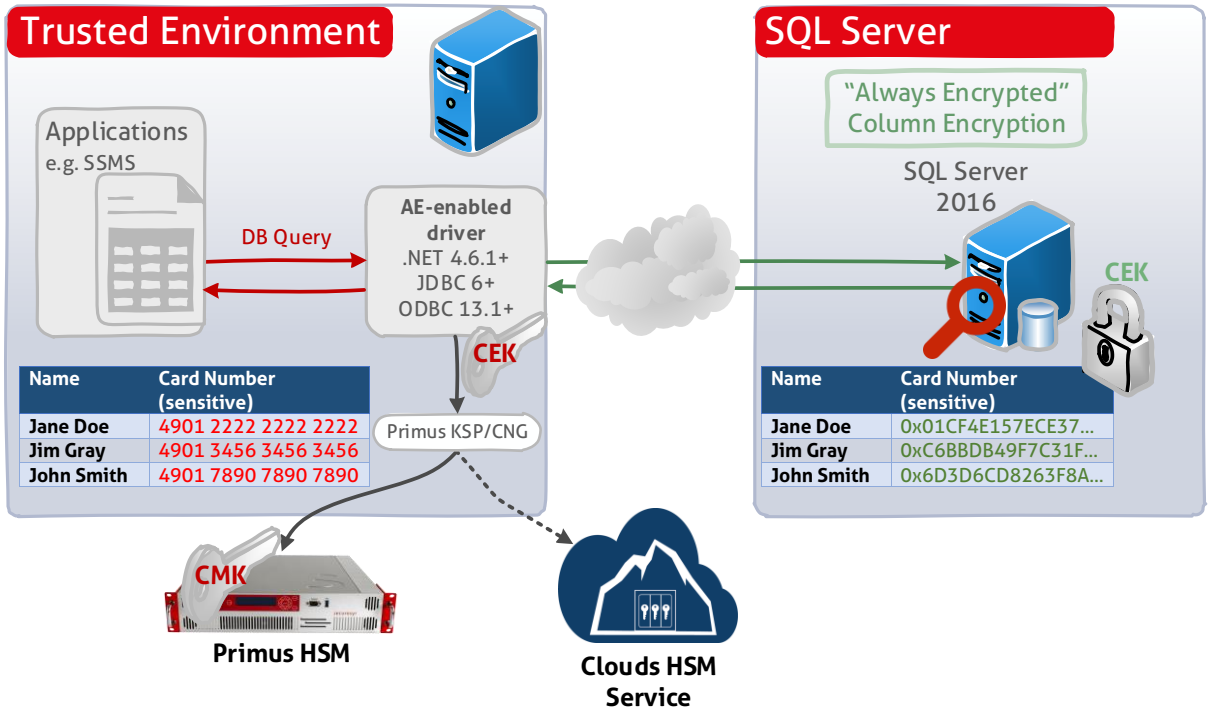
1 Introduction

This document describes how to secure private keys used by the Microsoft SQL Server by using the Securosys Primus HSM or Securosys Clouds HSM service.

Microsoft SQL Server 2016 (all editions) provides a new feature called "Always Encrypted" (AE), designed to protect sensitive data, such as credit card numbers or national identification numbers, both **at rest and in motion** between an on-premises client application and Azure or SQL Server databases.

There are a couple of good reasons why Always Encrypted should be used:

- **Regulatory support:** personal data must be protected by more and more industry regulations, e.g. General Data Protection Regulation (GDPR) or US GSA PII standard ("Personally Identifiable Information"). Not having appropriate technical and organizational protection mechanisms in place, the data owner can be severely penalized.
- **Data Security:** data always needed to be secure. AE brings another layer of data in motion protection, filling the gap if other transport security mechanisms are compromised (e.g. SSL)



The Primus HSMs are built to securely generate and store true random cryptographic keys, providing a central, certified secure storage. They also control and regulate access to the keys and the related cryptographic functionality. The Primus HSM combined with Microsoft SQL Server Always Encrypted meets or exceeds the best practice security requirements and is one step ahead of fulfilling your compliance demands by providing:

- Hardware-based secure generation of true random cryptographic keys
- Central and highly secure storage of cryptographic keys
- Load balancing and fail-over by clustering the HSMs

- Controlled and regulated access to the keys
- Hardware acceleration of cryptographic operations such as encryption, authentication, and digital signatures, relieving the host server of processor intensive computations
- Scalable performance at manageable cost

All certificate issuance and validation processes occur within the protected confines of the HSM. Private keys are never accessible outside the HSM.

The Primus HSM can easily be integrated in a Microsoft Windows system by installing the Primus CNG Provider. This enables all Windows servers and clients to generate and store their private keys and certificates securely in the HSMs, and perform all related cryptographic functionality, like signing or certificate validation, hardware accelerated on the Primus HSM.

1.1 Introduction to "Always Encrypted"

Data protected by Always Encrypted remains encrypted until it has reached the on–premises client application. This allows clients to encrypt sensitive data inside the client application and never reveal the encryption keys to the database engine. Therefore, it effectively mitigates man in the middle attacks and protects against unauthorized activity of rogue database administrators. AE provides a separation between those who own the data and those who manage the data.

Unlike Transparent Data Encryption (TDE), which encrypts the data and log files on disk but allows the data to be read by any application that queries the database, Always Encrypted requires your client application to use an AE-enabled driver to communicate with the database (currently Microsoft .NET Framework 4.6.1, JDBC 6, ODBC 13.1 or later).

Always Encrypted can be configured for individual database columns containing sensitive data, by defining the encryption algorithm and cryptographic keys used to protect the data in the column. There are two types of AE keys:

- A **Column Master Key (CMK)**, an asymmetric RSA encryption key (2048 bits), used to encrypt the Column Encryption Keys (CEK). The CMK can be protected by Hardware Security Modules and Clouds HSM (HSM as a service), using the CNG/KSP Provider.
- One or more **Column Encryption Keys (CEK)**, a symmetric AES key (256 bits), used to encrypt database columns.

The database engine stores encryption configuration for each column in database metadata. It also stores encrypted values of CEKs and the information about the location of CMKs, which are stored in external trusted key stores, such as hardware security modules, Windows Certificate Store on a client machine or Azure Key Vault.

Always Encrypted supports two types of 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, Yes/No etc.

- **Randomized** encryption: is more secure as it produces different cipher text values from the same plaintext every time the data is encrypted. However, prevents searching, grouping, indexing, and joining on encrypted columns.

The initial setup of Always Encrypted in a database involves generating AE keys, creating key metadata, configuring encryption properties of selected database columns, and/or encrypting data that may already exist in columns that need to be encrypted.
Please note: the database engine cannot be involved in key provisioning (AE keys) or data encryption/decryption operations. Some of the initial tasks are not supported in Transact-SQL and require the use of client-side tools (e.g. SQL Server Management Studio or PowerShell):

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

1.2 Requirements

Before you get started, ensure the following:

- A database
  - either Microsoft SQL Server 2016 SP1 (all editions) or later is installed
  - or Microsoft SQL Server 2016 Enterprise Edition
  - or Azure SQL Database V12
- On a client machine, hosting your client applications
  - Windows operating system with Crypto API Next Generation (CNG) (Server 2008 or higher, Windows Vista or higher).
  - Always Encrypted-enabled driver, currently
    - Microsoft .NET Framework 4.6.1 RC or later (e.g. for applications developed with Microsoft Visual Studio)
    - JDBC 6 or later
    - ODBC 13.1 or later
  - Securosys Primus CNG/KSP Provider installed and configured

1.2.1 Software and Hardware used for this Setup

The integration was performed and tested using the following configuration:

- Virtualization Software (VMWare Workstation 14.1.1)
- Microsoft Windows Server 2016/x64, with latest patches installed, hosting SQL Server 2016 SP1 (Standard edition)
- SQL Server Management Studio 17.5 (SSMS)

- Microsoft .NET Framework 4.7.1
- PowerShell version 5.1, PowerShellGet 1.6.0, PowerShell SqlServer 20.0
- Securosys Primus HSM CNG/KSP Provider V1.21.1
- Securosys Primus-X HSM V2.5.1

1.3 References and More Information

For more information about HSM administration, refer to the Primus HSM User Guide or contact Securosys support.

For more information about OS support, contact your Microsoft sales representative or integration partner.

The following references provide further information:

- Microsoft Always Encrypted (Database Engine) documentation https://docs.microsoft.com/en-us/sql/relational-databases/security/encryption/always-encrypted-database-engine
- Microsoft Always Encrypted (Client development) documentation https://docs.microsoft.com/en-us/sql/relational-databases/security/encryption/always-encrypted-client-development

The next edition of this application note will consider the topics role separation and key rotation.

2 Procedures

This procedure provides a straightforward integration process, which has been tested. Please take notice that there may be other ways to achieve interoperability. This guide assumes that you are familiar with the Primus HSM, Microsoft SQL Server and PowerShell.

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

Note: Throughout this guide, the term HSM refers to Securosys Primus HSM products.

2.1 Securosys Primus HSM Setup

Setting-up the Securosys Primus HSM hardware or your Clouds HSM partition is not part of this application note. Please refer to the corresponding Quick Start Guides and User Manuals.

2.2 SQL Server Setup

For a detailed installation procedure of SQL Server, Always Encrypted pre-requisites, and recommendations refer to the Microsoft SQL Server documentation.

2.3 Installing the Primus HSM CNG/KSP Provider on the Client Machines

The Primus CNG/KSP Provider must be installed on each Client machine running the Always Encrypted-enabled database applications.

Download and install the latest CNG/KSP provider from the Securosys support portal. Configure and test the Primus or Clouds HSM connections using the Securosys Key Storage Provider Configuration application.

For details regarding installation and configuration of the CNG/KSP provider consult the application note "PrimusHSM_CNGInstallation_AN", downloadable at the Securosys support portal.

2.4 Configuring "Always Encrypted" using SQL Server Management Studio (SSMS)

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

The following database permissions are required for Always Encrypted:

Kmgt[1]	Query[2]	Permission	Description
X		ALTER ANY COLUMN MASTER KEY	Required to create and delete a CMK
X		ALTER ANY COLUMN ENCRYPTION KEY	Required to create and delete a CEK
X	X	VIEW ANY COLUMN MASTER KEY DEFINITION	Required to access and read the metadata of the CMK to manage keys or query encrypted columns
X	X	VIEW ANY COLUMN ENCRYPTION KEY DEFINITION	Required to access and read the metadata of the CEK to manage keys or query encrypted columns.

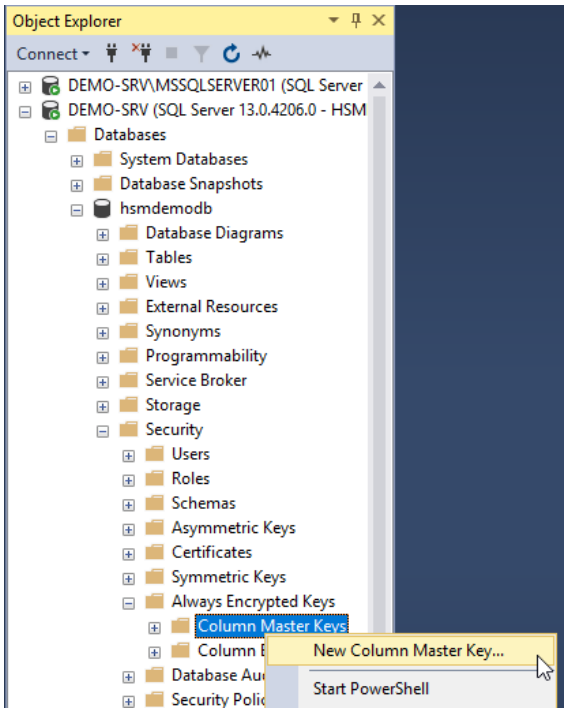
[1] Key management tasks by the Security Officer (creating/changing/reviewing DB key metadata)
[2] Required for querying encrypted columns, by the Database Administrator

See chapter 2.6 and the corresponding Microsoft documentation for applied restrictions.

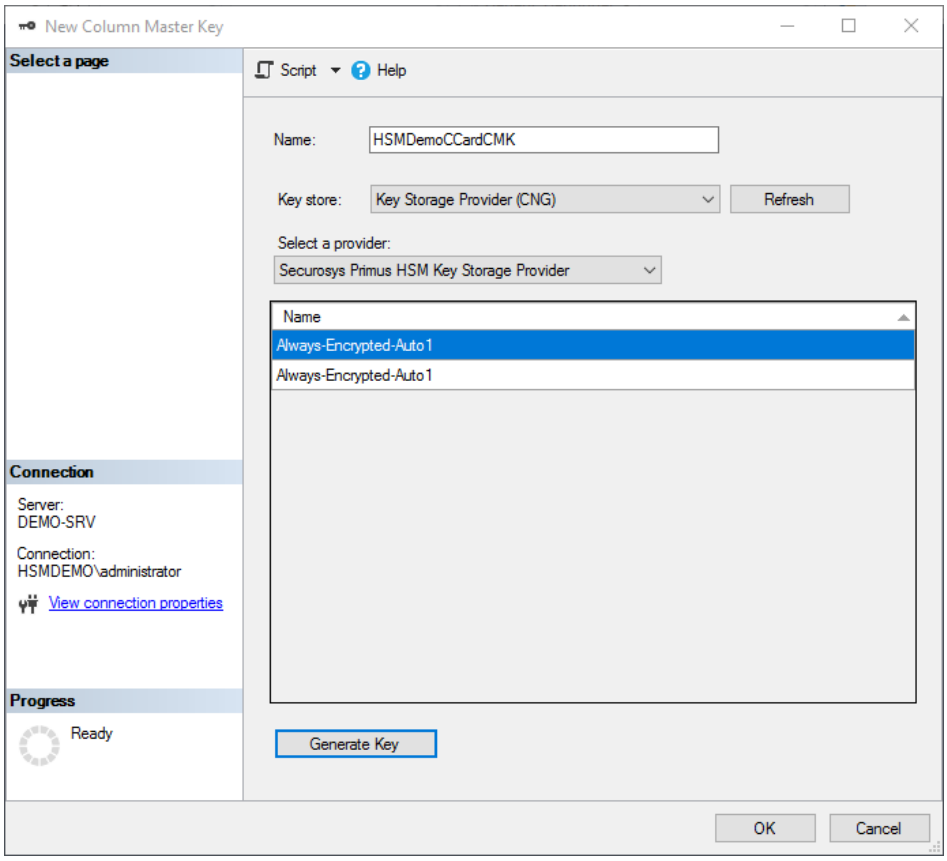
2.4.1 Creating the Column Master Key (CMK) using the Primus HSM CNG/KSP Provider

Start the Microsoft SQL Server Management Studio (SSMS) and connect to the desired database to protect. Create a Column Master Key (CMK) using the Primus HSM. This key will encrypt all subsequent Column Encryption Keys (CEK).

Within the SSMS, use the Object Explorer and select the [Security] folder under the desired database (in this example "hsmdemodb"). Click to expand [Always Encrypted Keys]. Right click on [Column Master Keys] and select [New Column Master Key...].



The "New Column Master Key" dialogue box will open.



- Enter a meaningful name for the CMK in the "Name" field.
- From the drop-down list select the "Key Storage Provider (CNG)" option and choose the "Securosys Primus HSM Key Storage Provider".

- Click [Generate Key] to generate a new key pair on the HSM.
- Click [OK] to finish the process.

On the HSM the key is named by default "Always-Encrypted-Auto1", using RSA 2048 bit, not exportable, key usage for decrypt and signing.

Now you have a Column Master Key called "HSMDemoCCardCMK", protected by the Primus HSM or Clouds HSM.

2.4.2 Creating the Column Encryption Key (CEK)

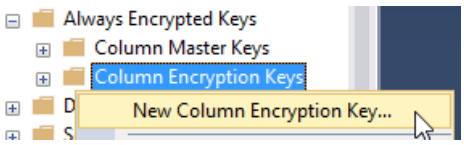
There are two approaches to generate the Column Encryption Key (CEK):

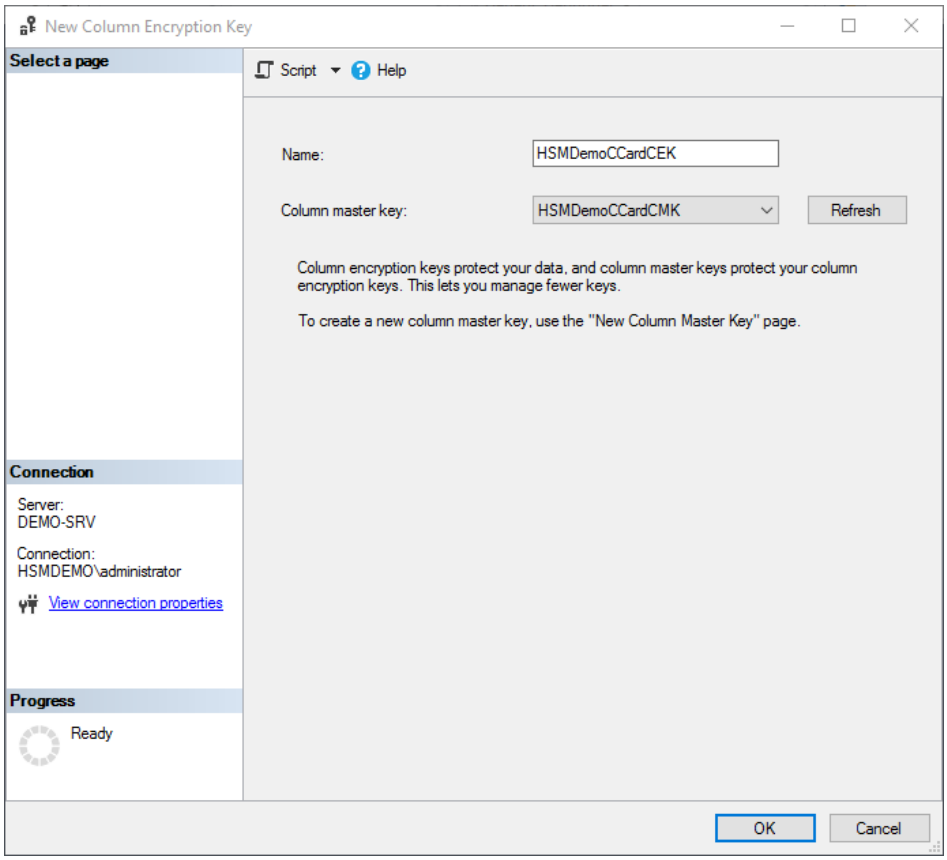
a) Defining them explicitly within the Security folder where you defined the CMK, having the advantage to name them manually

b) Generating them automatically when protecting columns using the Always Encrypted wizard (manual naming not possible).

The following described procedure is based on variant a).

Select [<Your_database>] [Security] [Always Encrypted Keys] [Column Encryption Keys]. Right click on "Column Encryption Keys" and select [New Column Encryption Key...].



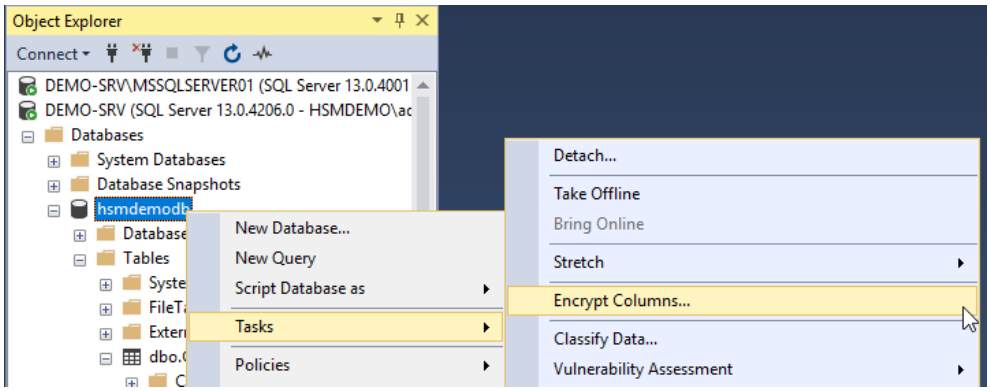


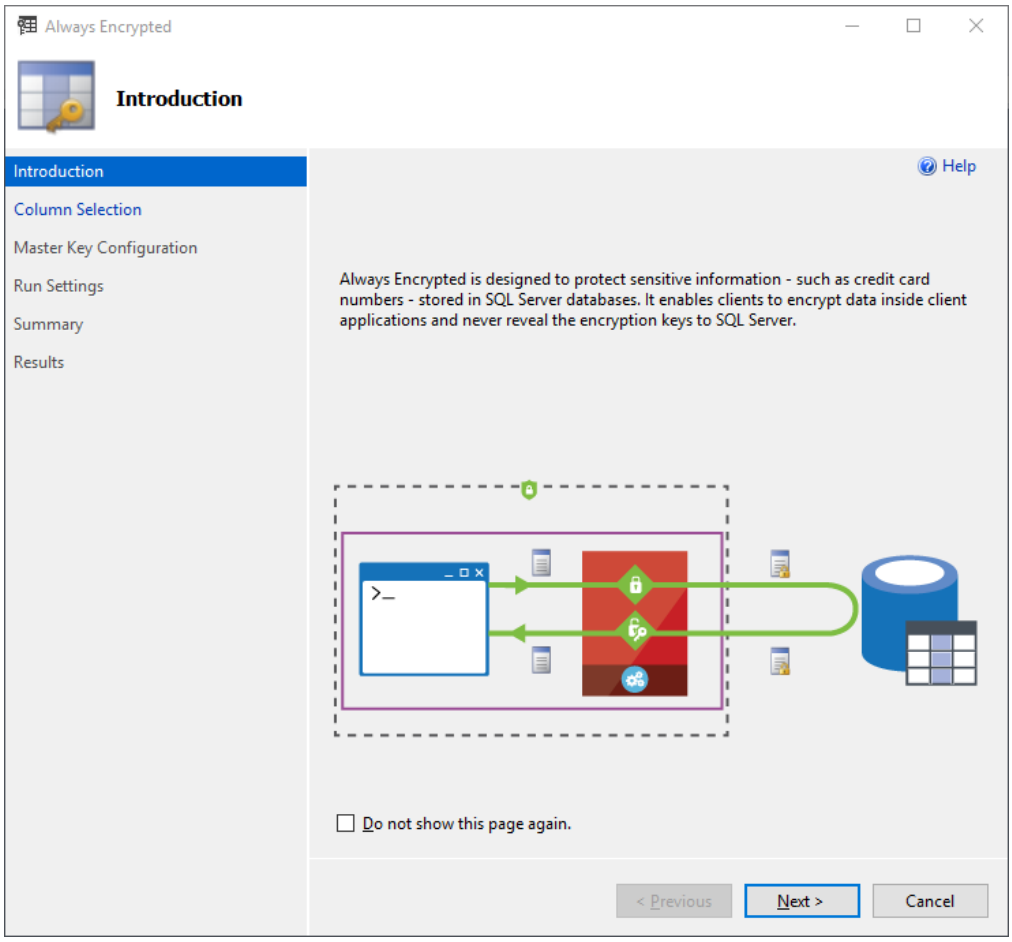
- Enter a meaningful name for the CEK in the Name field.
- Select your previously defined Column Master Key to protect the CEK.
- Click [OK] to finish the process.

Now you have a Column Encryption Key called "HSMDemoCCardCEK", protected by the Column Master Key "HSMDemoCCardCMK".

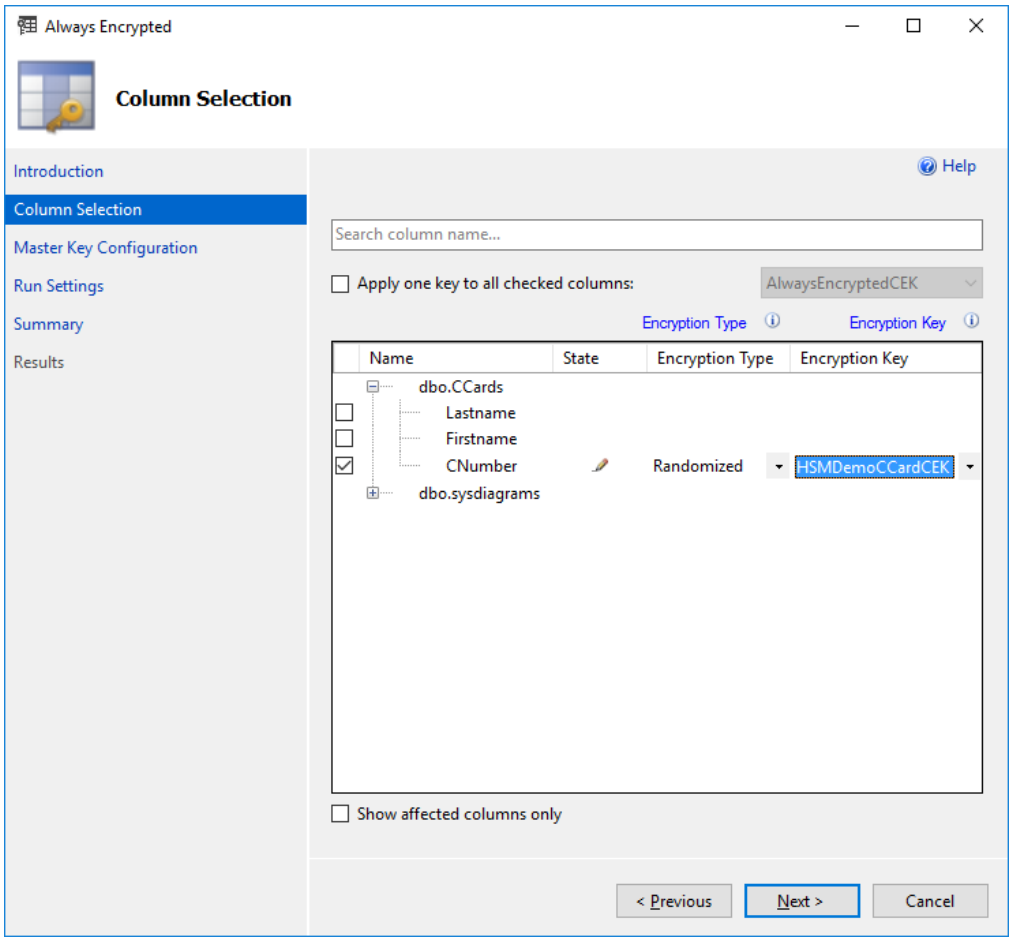
2.4.3 Enabling Column Encryption on the Database Table

To Enable Always Encrypted (and generate a Column Encryption Key), right click on the required database ("HSMDemoDB" in this example) and select [Tasks] [Encrypt Columns...] to open the Always Encrypted wizard.



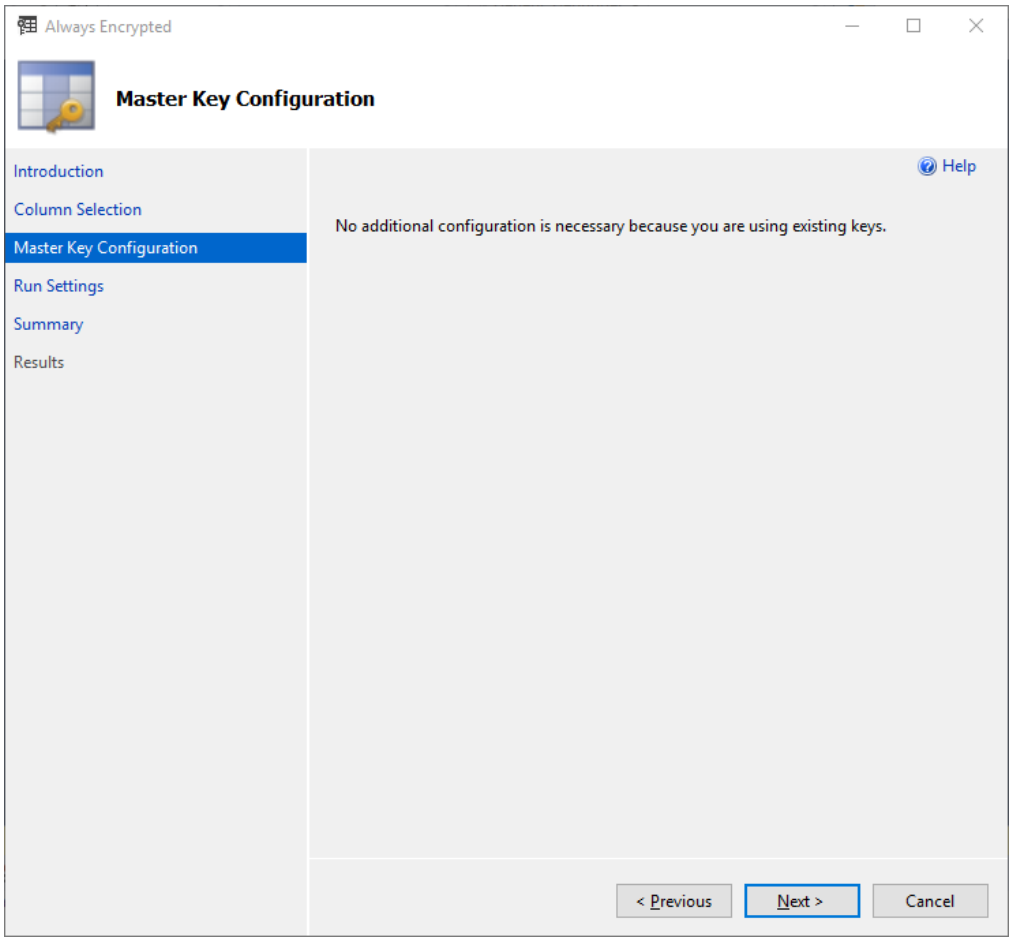


- Click [Next >] to skip the Introduction page.

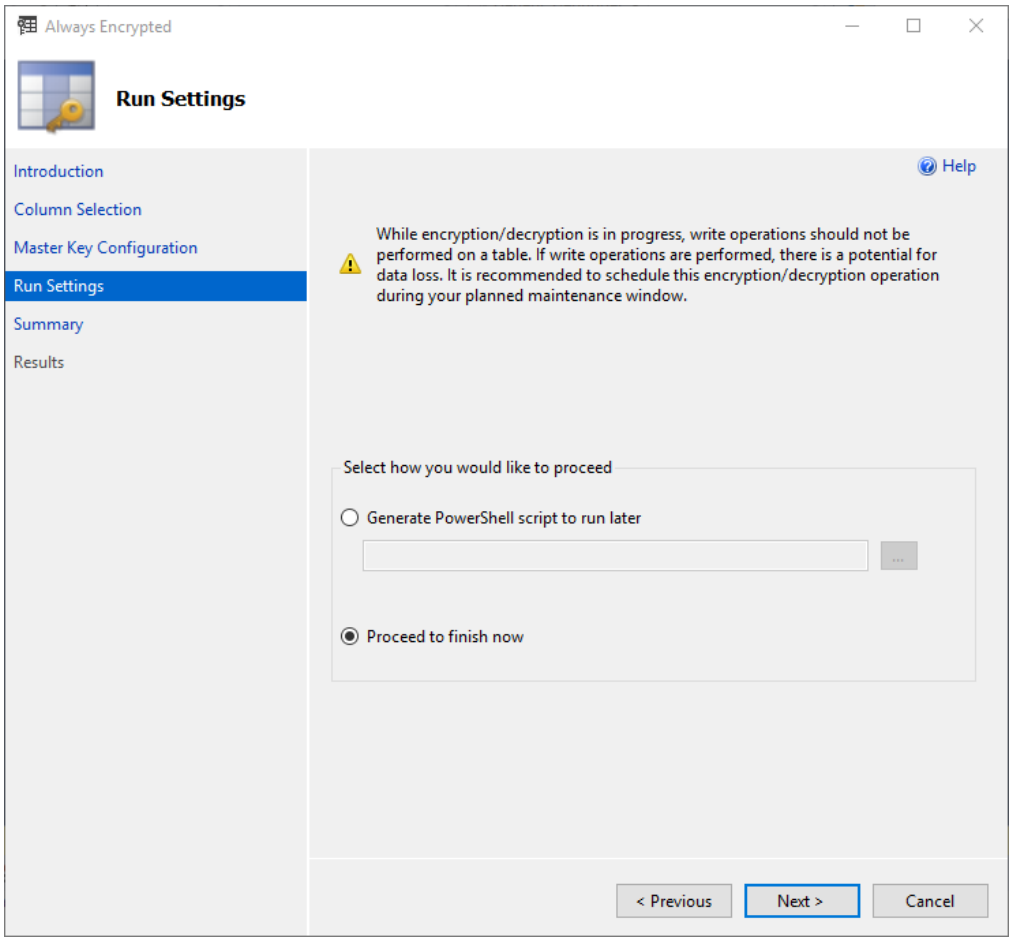


The Column Selection screen allows to specify the columns to encrypt, the encryption type and the Column Encryption Key.

- Under "Encryption Type" choose the encryption method from the drop-down box:
  - Deterministic
    "Uses a method which always generates the same encrypted value for any given plain text value. Using deterministic encryption allows grouping, filtering by equality, and joining tables based on encrypted values, but can also allow unauthorized users to guess information about encrypted values by examining patterns in the encrypted column. This weakness is increased when there is a small set of possible encrypted values, such as True/False, Yes/No etc. Deterministic encryption must use a column collation with a binary2 sort order for character columns."
  - Randomized
    "Uses a method that encrypts data in a less predictable manner. Randomized encryption is more secure, but prevents equality searches, grouping, indexing, and joining on encrypted columns."
  - Plaintext
    Only available to revert encrypted columns to an unencrypted state.
- Select your previously defined Column Encryption Key. If it was not pre-defined, then you can use an automatically named CEK (e.g. "CEK_Auto1 (New)"), generated in the next step.
- If all necessary columns are defined properly, click [Next >] to continue.



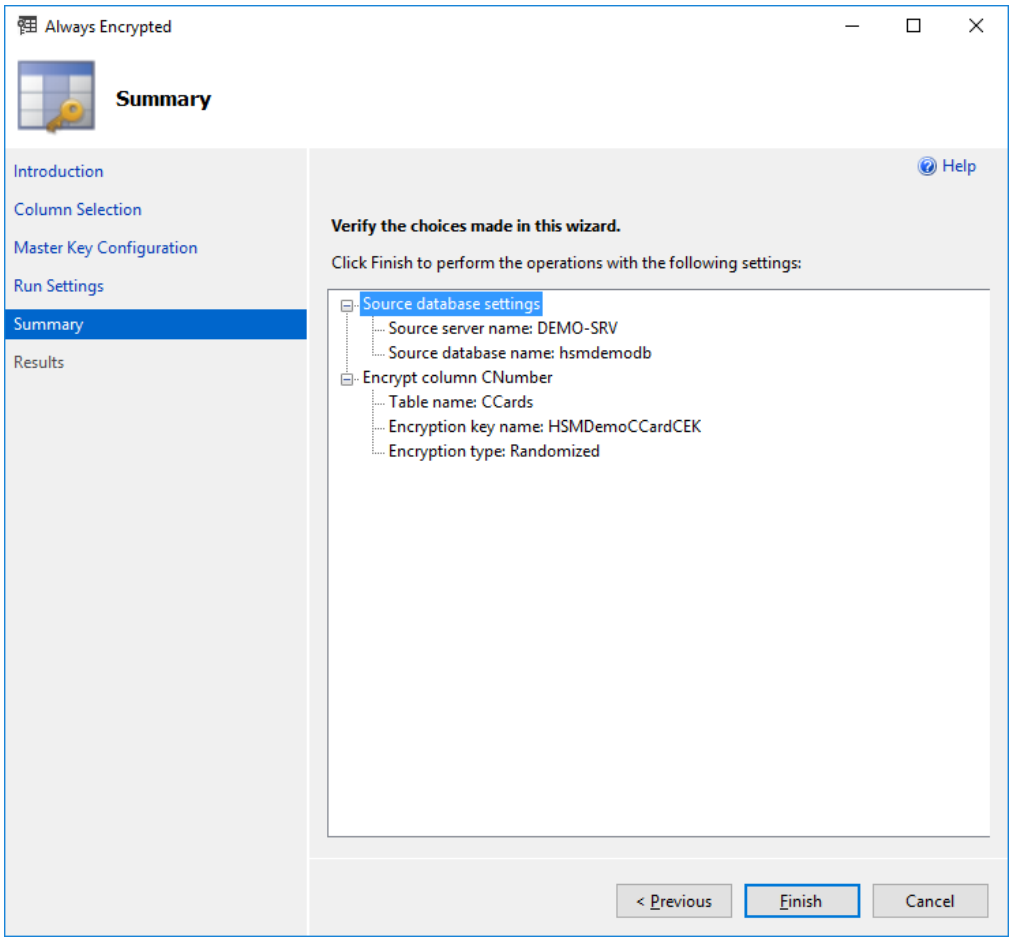
- On the Master Key Configuration page:
  - Using existing CEK: no additional configuration necessary
  - Generating a new CEK: make sure that you select the proper CMK (e.g. generated using the Primus CNG/KSP Provider)
- Click [Next >] to continue.



The process of encrypting database records can take a considerable amount of time, depending on the size and quantity of data. To mitigate the possibility of data corruption occurring as records are encrypted whilst being updated, it is advisable to back up the database and to only perform this activity when the database is off-line (schedule maintenance downtime).

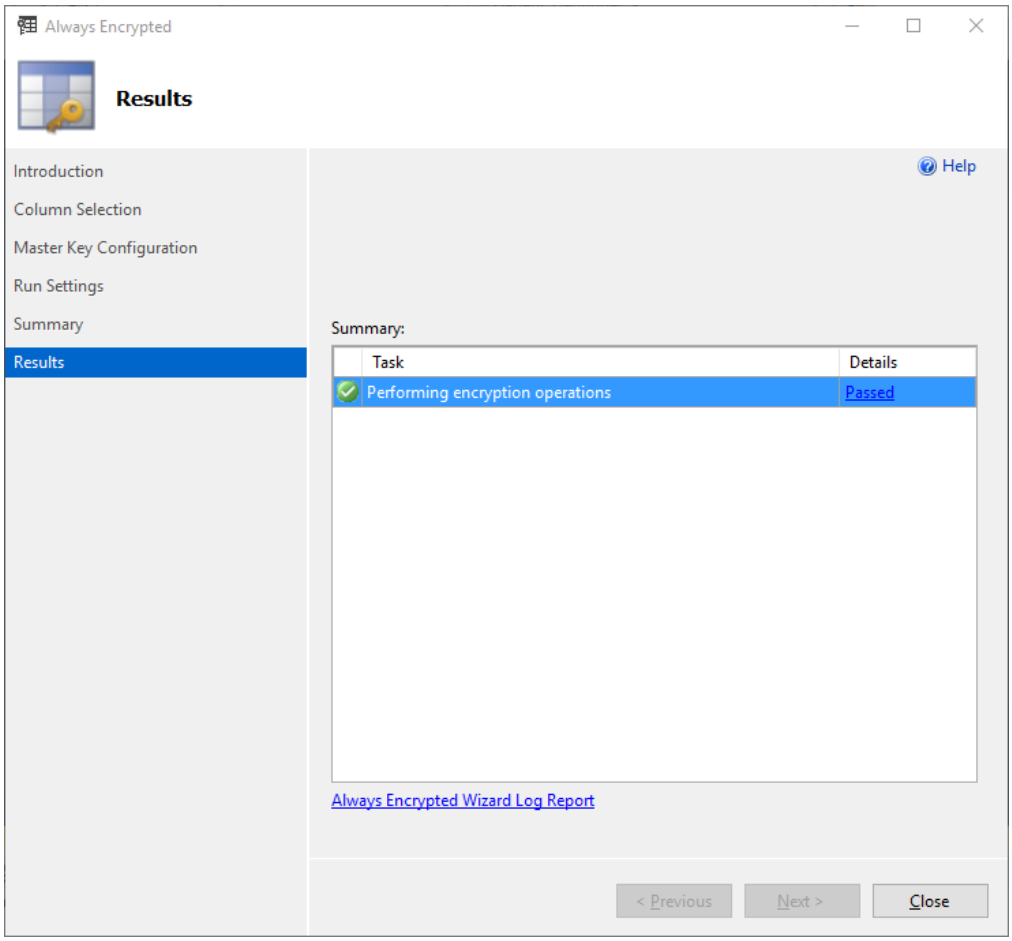
In this case we will continue and run the encryption straight away. Select the radio button, "Proceed to finish now" this will begin creating the CEK if necessary and using it to encrypt the specified columns in the database.

- Click [Next >] to view the Summary page.



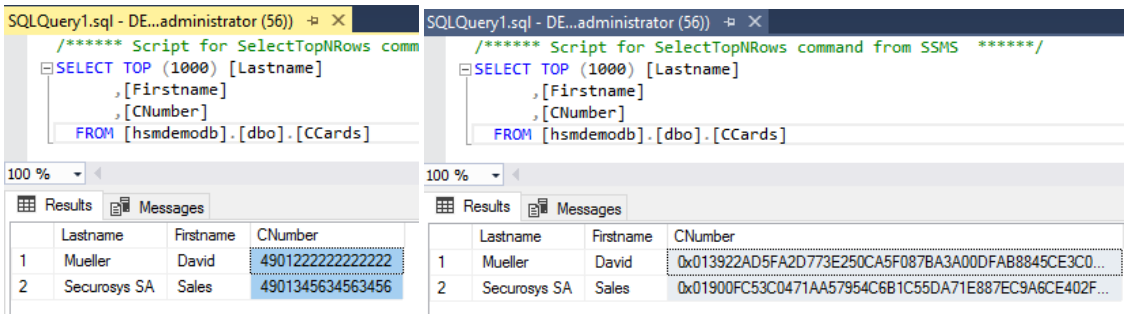
This page allows you to verify your configuration choices.

- Click [Next >] to view the Results page.



- Click [Close] to exit the Always Encrypted wizard.

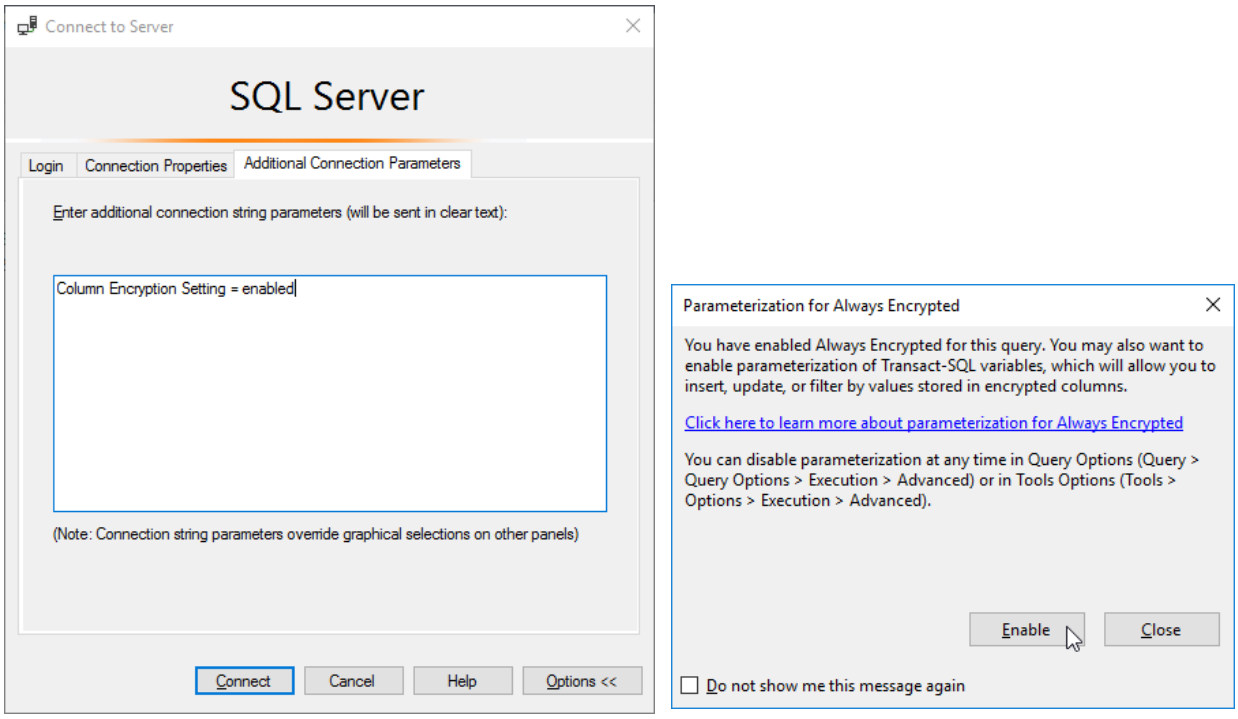
If you now open the table by right-clicking on the database table and selecting [Select Top 1000 Rows] you will see that the columns selected for encryption now appear as ciphertext:



2.4.4 Connecting to a Column Encrypted Database

Additional database connection parameters are required to show encrypted columns in plaintext.

The parameter is entered at the "Connect to Database Engine" logon screen. Select the required server name and click on [Options>>]. Select the "Additional Connection Parameters" tab and add the connection string "Column Encryption Setting = enabled" (without parenthesis "") and then click [Connect].



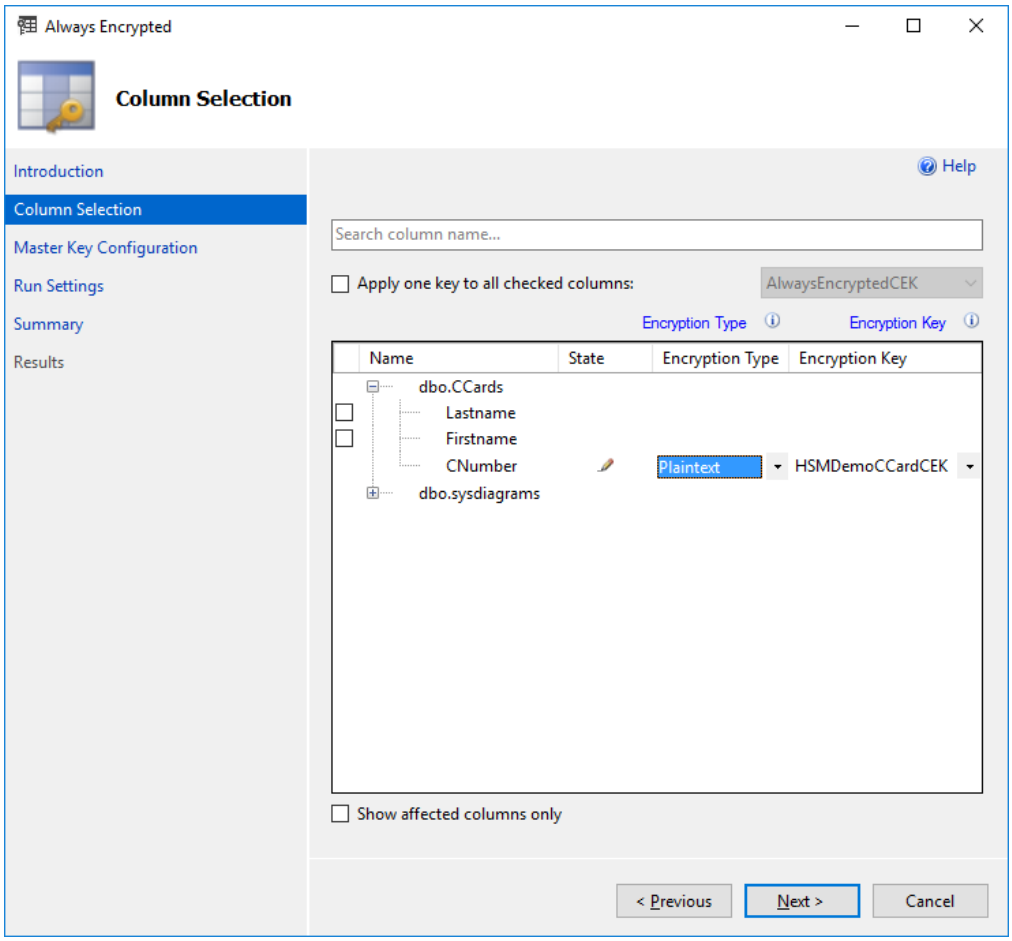
The first time you query the database you may get the "Parameterization for Always Encrypted" prompt. Click on [Enable] to proceed.

Running queries on the table will now show the original values decrypted by the CEK.

2.4.5 Removing Column Encryption on the Database Table

Use the SQL Server Management Studio to remove the protection provided by Always Encrypted column encryption.

Right click on the required database. Select [Tasks] [Encrypt Columns...] to open the Always Encrypted wizard.

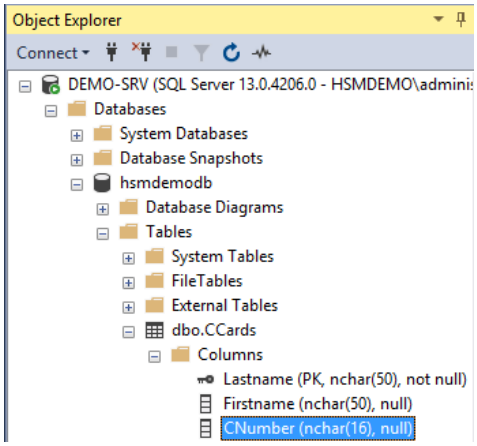


- On the Column Selection page change the Encryption type to "Plaintext" for the columns to decrypt.
- If all necessary columns are defined properly, click [Next >] [Next >] [Next >] and review on the Summary page that the correct Decrypt columns are listed.
- Click [Finish] to see the Results page and click [Close].

When you next log into the database you can remove the "Column Encryption Setting = enabled" string from the "Additional Connection Parameters" tab of the database login screen.

2.5 Configuring "Always Encrypted" using Power Shell

The following examples use the shown database "hsmdemodb" and the credit cards table "CCards", where the column "CNumber" should be AE column encrypted.



Note: Adapt the **red** values in the following context according your deployment.

2.5.1 Install and Configure SQLServer PowerShell Module

"Previous versions of the SqlServer module were included with SQL Server Management Studio (SSMS), but only with the 16.x versions of SSMS. To use PowerShell with SSMS 17.0 and later, the SqlServer module must be installed from the PowerShell Gallery."

Before updating PowerShellGet or PackageManagement, install the latest Nuget provider. Open a PowerShell (or PowerShell ISE) session as Administrator and run:

```
Install-PackageProvider Nuget –Force –Verbose
```

Update PowerShellGet:

```
Install-Module –Name PowerShellGet –Force –Verbose
```

Download and install the SqlServer module to configure Always Encrypted using Power Shell:

```
Install-Module -Name SqlServer -Force -Verbose -AllowClobber
```

Note: The "-AllowClobber" parameter allows you to import the specified commands if it exists in the current session.

If you are using PowerShell ISE refresh the Commands pane, if you are using PowerShell open a new session.

Check the install by running:

```
Get-Module -list -Name SqlServer

    Directory: C:\Program Files\WindowsPowerShell\Modules

ModuleType Version    Name                                ExportedCommands
---------- -------    ----                                ----------------
Script     21.0.17224 SqlServer                           {Add-SqlFirewallRule, ConvertFrom-….
Manifest   20.0       SqlServer                           {Add-SqlColumnEncryptionKeyValue, ….
```

2.5.2 Creating the Column Master Key (CMK) using the Primus HSM CNG/KSP Provider

Generate an RSA key pair via the Primus CNG/KSP provider, for use as a Column Master Key:

```
$cngProviderName = "Securosys Primus HSM Key Storage Provider"
$cngAlgorithmName = "RSA"
$cngKeySize = 2048 # Recommended key size for Always Encrypted CMK
$cngKeyName = "AlwaysEncryptedCMK" #Name identifying your new key in the KSP
$cngProvider = New-Object System.Security.Cryptography.CngProvider($cngProviderName)
$cngKeyParams = New-Object System.Security.Cryptography.CngKeyCreationParameters
$cngKeyParams.provider = $cngProvider
$cngKeyParams.KeyCreationOptions =
[System.Security.Cryptography.CngKeyCreationOptions]::OverwriteExistingKey
$keySizeProperty = New-Object System.Security.Cryptography.CngProperty("Length",
[System.BitConverter]::GetBytes($cngKeySize),
[System.Security.Cryptography.CngPropertyOptions]::None);
$cngKeyParams.Parameters.Add($keySizeProperty)
$cngAlgorithm = New-Object System.Security.Cryptography.CngAlgorithm($cngAlgorithmName)
$cngKey = [System.Security.Cryptography.CngKey]::Create($cngAlgorithm, $cngKeyName, $cngKeyParams)
```

The above example generates a 2048-bit RSA key pair named "AlwaysEncryptedCMK", protected by the Primus HSM.

Invoke the "**New-SqlCngColumnMasterKeySettings**" cmdlet to create a SqlColumnMasterKeySettings object, describing an asymmetric key stored in a key store supporting the CNG API:

```
## Specify the Column Master Key settings for importing into the database:
$CmkSettings = New-SqlCngColumnMasterKeySettings -CngProviderName "Securosys Primus HSM Key Storage
Provider" -KeyName "AlwaysEncryptedCMK"
```

Then create the Column Master Key object in the database:

```
New-SqlColumnMasterKey "AlwaysEncryptedCMK" -ColumnMasterKeySettings $CmkSettings -Path
SQLSERVER:\SQL\server_name\DEFAULT\Databases\your_database
```

E.g. using our example database "hsmdemodb":

```
New-SqlColumnMasterKey "AlwaysEncryptedCMK" -ColumnMasterKeySettings $CmkSettings -Path
SQLSERVER:\SQL\DEMO-SRV\DEFAULT\Databases\hsmdemodb
```

2.5.3 Creating the Column Encryption Key (CEK)

Once the Column Master Key has been successfully generated create a Column Encryption Key object in the database:

```
New-SqlColumnEncryptionKey -Name "AlwaysEncryptedCEK" -ColumnMasterKeyName "AlwaysEncryptedCMK" –Path
SQLSERVER:\SQL\server_name\DEFAULT\Databases\your_database
```

The resulting Column Encryption Key (CEK) is a 256-bit symmetric key protected by the Column Master Key (CMK).

2.5.4 Encrypting Columns using the Column Encryption Key

Open a PowerShell (or PowerShell ISE) session with elevated permissions (right click and select "Run as Administrator") and run the following to encrypt a given column in the specified database. Adjust the **red** highlighted values to those suitable for your database name and data columns that you want to encrypt.
Remember, the "EncryptionType" values are one of either: Deterministic, Randomized, Plaintext.

```
# Import Module SqlServer
Import-Module SqlServer
```

```
# Set up connection and database SMO objects
$sqlConnectionString = "Data Source=server_name;Initial Catalog=your_database;Integrated
Security=True;MultipleActiveResultSets=False;Connect
Timeout=30;Encrypt=False;TrustServerCertificate=False;Packet Size=4096;Application Name=`"Microsoft
SQL Server Management Studio`""
$smoDatabase = Get-SqlDatabase -ConnectionString $sqlConnectionString

# Change encryption schema
$encryptionChanges = @()

# Add changes for table [dbo].[DemoTable]
$encryptionChanges += New-SqlColumnEncryptionSettings -ColumnName dbo.DemoTable.ColToEncrypt
-EncryptionType Randomized -EncryptionKey "AlwaysEncryptedCEK"
Set-SqlColumnEncryption -ColumnEncryptionSettings $encryptionChanges -InputObject $smoDatabase
```

E.g. using our example database "hsmdemodb":

```
# Import Module SqlServer
Import-Module SqlServer

# Set up connection and database SMO objects
$sqlConnectionString = "Data Source=DEMO-SRV;Initial Catalog=hsmdemodb;Integrated
Security=True;MultipleActiveResultSets=False;Connect
Timeout=30;Encrypt=False;TrustServerCertificate=False;Packet Size=4096;Application Name=`"Microsoft
SQL Server Management Studio`""
$smoDatabase = Get-SqlDatabase -ConnectionString $sqlConnectionString

# Change encryption schema
$encryptionChanges = @()

# Add changes for table [dbo].[hsmdemodb]
$encryptionChanges += New-SqlColumnEncryptionSettings -ColumnName dbo.CCards.CNumber -EncryptionType
Randomized -EncryptionKey "AlwaysEncryptedCEK"
Set-SqlColumnEncryption -ColumnEncryptionSettings $encryptionChanges -InputObject $smoDatabase
```

2.5.5 Removing Column Encryption

To remove column encryption from previously encrypted column data, replace the column EncryptionType value with "Plaintext" and execute Set-SqlColumnEncryption cmdlet.

Take the database preferably offline before proceeding to remove column encryption.

```
# Import Module SqlServer
Import-Module SqlServer

# Set up connection and database SMO objects
$sqlConnectionString = "Data Source=server_name;Initial Catalog=your_database;Integrated
Security=True;MultipleActiveResultSets=False;Connect
Timeout=30;Encrypt=False;TrustServerCertificate=True;Packet Size=4096;Application Name=`"Microsoft
SQL Server Management Studio`";Column Encryption Setting=Enabled"
$smoDatabase = Get-SqlDatabase –ConnectionString $sqlConnectionString

# Change encryption schema
$encryptionChanges = @()

# Add changes for table [dbo].[DemoTable]
$encryptionChanges += New-SqlColumnEncryptionSettings –ColumnName dbo.DemoTable.ColToDecrypt
–EncryptionType Plaintext
Set-SqlColumnEncryption –ColumnEncryptionSettings $encryptionChanges –InputObject $smoDatabase
```

The "Always Encrypted" column data will revert to plaintext.

Remove the "Column Encryption Setting = enabled" string from the "Additional Connection Parameters" field of the database login screen.

Note: When removing Always Encryption from your database columns, ensure that all columns appear in plaintext. You must delete any Column Encryption Keys (CEK) before you can drop the Column Master Keys (CMK).

2.6 Always Encrypted – Feature Details

Always Encrypted imposes some restrictions, summarized below. For details consult the Microsoft website: https://docs.microsoft.com/en-us/sql/relational-databases/security/encryption/always-encrypted-database-engine

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

Always Encrypted is not supported for the columns with the below characteristics (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